The German Informatics society (GI) publishes this series in order

- to make available to a broad public recent findings in informatics (i.e. computer science and information systems)
- to document conferences that are organized in cooperation with GI and
- to publish the annual GI Award dissertation.

Broken down into the fields of "Seminars", "Proceedings", "Monographs" and "Dissertation Award", current topics are dealt with from the fields of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure the high level of the contributions.

The volumes are published in German or English

Information: http://www.gi-ev.de/LNI

Weikum, Schöning, Rahm (Hrsg.): BTW 2003

GI-Edition

Lecture Notes in Informatics

Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.)

BTW 2003 Datenbanksysteme für Business, Technologie und Web

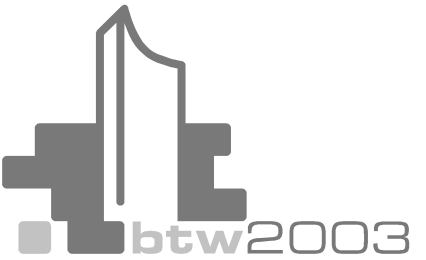
Tagungsband der 10. BTW-Konferenz
26.-28. Februar 2003, Leipzig

This volume contains the Proceedings of the Tenth Conference on Database Systems for Business, Technology, and Web, which took place in February 2003 in Leipzig. It includes three keynotes by internationally leading experts, 14 full and 17 short papers selected by the program committee, 2 short papers by the recipients of the Dissertation Award of the GI Section on Databases and Information Systems, as well as 11 papers from the industrial program. The topics span a wide spectrum from traditional database system issues to modern applications such as digital libraries or electronic process management as well as current and anticipated trends such as XML, Web services, peer-to-peer data management, or the Semantic Web research direction.

P-26

Proceedings

Gerhard Weikum
Harald Schöning
Erhard Rahm
(Herausgeber)



Datenbanksysteme für Business, Technologie und Web (BTW)

10. GI-Fachtagung
Leipzig, 26.-28. Februar

Herausgeber

Gerhard Weikum
Universität des Saarlandes
Im Stadtwald
D-66123 Saarbrücken
weikum@cs.uni-sb.de

Harald Schöning
Software AG
Uhlandstraße 12
D-64297 Darmstadt
Harald.Schoening@softwareag.com

Erhard Rahm
Universität Leipzig
Institut für Informatik
Augustusplatz 10-11
D-04109 Leipzig
rahm@informatik.uni-leipzig.de

Vorwort

Die 10. BTW-Tagung der Gesellschaft für Informatik (GI) findet vom 26. bis zum 28. Februar 2003 im traditionsreichen Leipzig statt, der Stadt von Johann Sebastian Bach und Auerbachs Keller, der Stadt der Völkerschlacht anno 1813 und der Stadt, in der die deutsche Wiedervereinigung mit den Montagsdemonstrationen im Herbst 1989 ihren Anfang nahm. Die BTW'2003 wird in den Räumen der Universität Leipzig am Augustusplatz ausgerichtet, direkt im Zentrum der Stadt. Die Universität Leipzig wurde vor nahezu 600 Jahren gegründet (1409) und verbindet Tradition und Innovation in besonderer Weise. Die Informatik wurde nach der Wende aufgebaut und hat sich mittlerweile an der Universität und in der deutschen Informatiklandschaft sehr gut etabliert.

Die BTW ist die zentrale Tagung des GI-Fachbereichs Datenbanken und Informationssysteme: Wissenschaftler, Praktiker und Anwender treffen sich alle zwei Jahre auf dieser Tagung. Die Tradition der BTW hat ihren Anfang in der ersten Tagung 1985 in Karlsruhe zu einer Zeit, in der sich Datenbanksysteme von den klassischen betrieblichen Einsatzfeldern zu Anwendungen in Büro, Technik und Wissenschaft entwickelten, daher der Name BTW. Heute ist Datenbanktechnologie der wichtigste Stützpfeiler der Softwarebranche generell; sie ist unverzichtbar für globale Business-Portals und elektronische Prozesse, als Infrastruktur in der Telekommunikation und anderen eingebetteten Technologien, als skalierbares Rückgrat digitaler Bibliotheken und vieler Data-Mining-Werkzeuge sowie für viele Arten von Web-Anwendungen. Im Zeitalter der Informationsexplosion, Virtualisierung und Globalisierung kommen auf die Datenbanktechnologie aber weiterhin neue Herausforderungen zu, wenn es etwa um die Informationsintegration aus heterogenen, verteilten Datenquellen, Internet-weites kollaboratives Data-Mining oder die Gestaltung der Vision eines "Semantic Web" geht. Die Veränderungen bewogen den GI-Fachbereich Datenbanken und Informationssysteme, die Abkürzung BTW neu zu definieren: ab der diesjährigen 10. Ausgabe steht der Name für Datenbanksysteme für Business, Technologie und Web.

Das wissenschaftliche Programm der BTW'2003 umfasst die eingeladenen Beiträge dreier international führender Forscher - Frank Leymann, Alon Halevy und Peter Lokkemann - sowie 14 Lang- und 17 Kurzbeiträge, die das Programmkomitee aus 78 Einreichungen ausgewählt hat. Das Industrieprogramm umfasst 7 Lang- und 4 Kurzbeiträge, die vom Industriekomitee aus den Einreichungen ausgewählt und teilweise aktiv angeworben wurden. Wie schon in früheren BTW-Jahren wird dieses Kernprogramm ergänzt durch die Datenbanktutorientage der Deutschen Informatikakademie, das Studierendenprogramm, bei dem hervorragende Diplom- und Studienarbeiten vorgestellt werden, zwei Workshops zu den Themen "Datenbanken und E-Learning" und "Knowledge Discovery" sowie die Verleihung der von der Firma IBM gestifteten Dissertationspreise des GI-Fachbereichs Datenbanken und Informationssysteme.

In Leipzig findet die BTW erstmals zeit- und ortsgleich mit einer zweiten großen Informatiktagung, der 13. GI-Fachtagung über Kommunikation in verteilten Systemen, kurz KiVS, statt. BTW und KiVS starten gemeinsam mit der feierlichen Eröffnung im Gewandhaus und dem ersten eingeladenen Vortrag über Web-Services. Wir hoffen, dass dieses Bündnis zweier traditionsreicher Konferenzen dazu ermutigt, über die Grenzen des engeren Fachgebiets zu schauen und den Blick für die spannenden Verflechtungen und potentiellen Synergien zwischen Kommunikations- und Datenbanktechnologien zu schärfen. Ein weiteres Novum in der BTW-Geschichte ist, dass es erstmals zusätzlich zu dem Ihnen vorliegenden Tagungsband in der LNI-Reihe der GI einen elektronischen Tagungsband gibt, der direkt vom Trierer DBLP-Server erreichbar ist. Wir hoffen, dass diese zeitgemäße Neuerung zusammen mit dem zunehmenden Trend, Beiträge in englischer Sprache zu verfassen, die internationale Sichtbarkeit der BTW noch weiter verbessert.

Zum Gelingen einer Konferenz tragen viele bei. Wir möchten uns herzlich bedanken bei allen Vortragenden, insbesondere den eingeladenen Rednern, und ihren Koautoren, sowie bei denjenigen, die Beiträge eingereicht haben, auch wenn dann vielleicht das letzte bisschen Fortune bei der sehr selektiven Auswahl gefehlt hat. Ebenso herzlich danken wir den Mitgliedern des Programmkomitees und des Industriekomitees sowie den zusätzlichen externen Gutachtern für ihre Sorgfalt und Mühe, sowie den Mitgliedern des Organisationskomitees und den Verantwortlichen für Tutorien, Studierendenprogramm, Dissertationspreisen und Workshops. Besonderer Dank gebührt Ulrike Greiner, Thomas Kudraß, Katrina Leyking und Carsten Simon, die den Tagungsleiter und den Programmkomiteevorsitzenden mit großem Engagement organisatorisch unterstützt haben. Last but not least möchten wir den Sponsoren der diesjährigen BTW-Tagung für die finanzielle Unterstützung sehr herzlich danken. Es ist sehr beachtlich und erfreulich, dass es gerade in den schwierigen Zeiten der aktuellen Rezession gelungen ist, beim Sponsoring ein Rekordergebnis in der Geschichte der BTW zu erzielen.

Wir begrüßen Sie, liebe Tagungsteilnehmer, herzlich zur BTW'2003 in Leipzig und hoffen, dass Sie Vorträge und Rahmenprogramm spannend und erkenntnisreich finden. Um die in wenigen Tagen beginnende Zeit bis zur nächsten BTW-Konferenz 2005 zu überbrücken, empfehlen wir eingefleischten BTW-Fans und denjenigen, die es mit der Leipziger Konferenz hoffentlich werden, die Web-Seite mit der URL http://www.btw-konferenz.de auch in Zukunft häufiger zu besuchen.

Erhard Rahm, Tagungsleiter
Harald Schöning, Vorsitzender des Industriekomitees
Gerhard Weikum, Vorsitzender des Programmkomitees

Tagungsleitung:

Erhard Rahm, Universität Leipzig

Programmkomitee:

Gerhard Weikum (Vorsitz), Universität des Saarlandes
Hans-Jürgen Appelrath, Universität Oldenburg
Thomas Brinkhoff, FH Oldenburg
Stefan Conrad, Ludwig-Maximilian-Universität München
Stefan Deßloch, Universität Kaiserslautern
Klaus Dittrich, Universität Zürich
Peter Fankhauser, Fraunhofer IPSI und Infonyte, Darmstadt
Norbert Fuhr, Universität Duisburg
Torsten Grust, Universität Konstanz
Andreas Heuer, Universität Rostock
Stefan Jablonski, Universität Erlangen
Gerti Kappel, Technische Universität Wien
Alfons Kemper, Universität Passau
Arnd Christian König, Microsoft Research, Redmond
Donald Kossmann, Technische Universität München
Thomas Kudraß, HTWK Leipzig
Klaus Küspert, Universität Jena
Alexander Maedche, FZI Karlsruhe
Volker Markl, IBM Almaden Research Center, San Jose
Bernhard Mitschang, Universität Stuttgart
Erhard Rahm, Universität Leipzig
Kai Rannenberg, Universität Frankfurt
Manfred Reichert, Universität Ulm
Norbert Ritter, Universität Hamburg
Kai-Uwe Sattler, Univ. Magdeburg
Bernhard Seeger, Universität Marburg
Agnes Voisard, Freie Universität Berlin
Gottfried Vossen, Universität Münster
Mechtild Wallrath, DZ-Bank AG, Karlsruhe
Walter Waterfeld, Software AG
Roger Weber, ETH Zürich
Mathias Weske, Hasso-Plattner-Institut Potsdam
Robert Winter, Universität St. Gallen

Industriekomitee:

Harald Schöning (Vorsitz), Software AG
Achim Kraiß, SAP
Frank Leymann, IBM
Thomas Ruf, GfK Marketing Services
Uta Störl, Dresdner Bank

Studierendenprogramm:

Andreas Oberweis, Universität Frankfurt
Gunter Saake, Universität Magdeburg

Tutorienprogramm:

Klaus Küspert, Universität Jena

Organisationskomitee:

Erhard Rahm, Universität Leipzig
Timo Böhme, Universität Leipzig
Ulrike Greiner, Universität Leipzig
Thomas Kudraß, HTWK Leipzig
Dieter Sosna, Universität Leipzig
Myra Spiliopoulou, HHL Leipzig

Leitungsgremium des GI-Fachbereichs Datenbanken und Informationssysteme und Gutachter für die Dissertationspreise:

Hans-Jürgen Appelrath, Universität Oldenburg
Stefan Conrad, Ludwig-Maximilian-Universität München
Peter Dadam, Universität Ulm
Klaus Dittrich, Universität Zürich
Nobert Fuhr, Universität Duisburg
Theo Härder, Universität Kaiserslautern
Andreas Heuer, Universität Rostock
Klaus Küspert, Universität Jena
Andreas Oberweis, Universität Frankfurt
Erhard Rahm, Universität Leipzig
Gunter Saake, Universität Magdeburg
Gottfried Vossen, Universität Münster
Mechtild Wallrath, DZ-Bank AG, Karlsruhe
Gerhard Weikum, Universität des Saarlandes
Mathias Weske, Hasso-Plattner-Institut Potsdam

Weitere Gutachter

Thomas Aden
Evguenia Altareva
Philipp Bender
Bettina Berendt
Martin Bernauer
Timo Böhme
Dietrich Boles
Ilvio Bruder
Zhiyuan Chen
Michael Christoffel
Suo Cong
Peter Dadam
Daniela Damm
Hai Hong Do
Anca Dobre
Ruxandra Domenig
Stefan Figge
Michael Gesmann
Christoph Gollmick
Marco Grawunder
Ulrike Greiner
Farshad Hakimpour
Arne Harren
Christoph Hartwich
Axel Herbst
Holger Hinrichs
Thomas B. Hodel
Andreas Hotho
Jens Hündling
Markus Keidl
Maurice van Keulen
Toralf Kirsten
Thomas Klement
Meike Klettke
Birgitta König-Ries
Frank Köster
Gerhard Kramler
Jens Lechtenbörger
Wolfgang Lehner
Hans-Peter Leidhold
Jens Lufter
Eitel von Maur
Christian Meiler
Jürgen Meister
Sergey Melnik
Holger Meyer
Boris Motik
Jutta Mülle
Robert Müller
Felix Naumann
Frank Oldenettel
Ilia Petrov
Denny Priebe
Andreas Rann
Ralf Rantzau
Joachim Schelp
Jürgen Schlegelmilch
Frank Schön
Gregor Schrott
Hilmar Schuschel
Andre Seifert
Stefan Seltzsam
Markus Sinnwell
Steffen Skatulla
Marko Smiljanic
Günther Specht
Bernhard Stegmaier
Knut Stolze
Heiko Tapken
Thomas Tesch
Raphael Volz
Gunnar Weber
Christian Wiesner
Xiangru Yuan
Bernhard Zeller
Andre Zeitz
Patrick Ziegler

Sponsoren


IBM

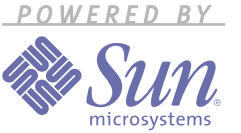

ORACLE


software AG
THE XML COMPANY


SYBASE


Microsoft
Research


InterSystems
CACHE
Make Applications Faster


POWERED BY
Sun
microsystems


forcont
business technology gmbh


transaction
software
intelligence in databases


robotron


infonyte


Cronide
Soft AG


datafactory
AKTIENGESELLSCHAFT

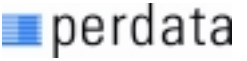

sdm

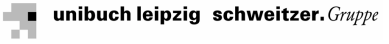

Sparkasse
Leipzig


BI Business Intelligence


Pearson
Studium


adesso


perdata


unibuch leipzig schweitzer. Gruppe


dpunkt.verlag


LIV


UNIVERSITÄT LEIPZIG


HTWK


HHL


UNIVERSITÄT
DES
SAARLANDES

BTW-Dissertationspreis
gestiftet von der IBM Deutschland GmbH

Die bei der BTW'2001 in Oldenburg erstmals durchgeführte Auszeichnung hervorragender Dissertationen im Gebiet der Datenbanken und Informationssysteme wird in Leipzig dank des erneuten finanziellen Engagements der Firma IBM fortgeführt. Für das Auswahlverfahren wurden nur Arbeiten akzeptiert, die im Zeitraum 10/2000 bis 9/2002 mit der Promotionsprüfung abgeschlossen wurden. Die Kandidatinnen und Kandidaten wurden von ihren betreuenden Professoren nominiert, eine Selbstbewerbung war nicht möglich. Gutachter über die eingereichten Arbeiten waren die Mitglieder des Leitungsgremiums des GI-Fachbereichs Datenbanken und Informationssysteme unter der Leitung von Theo Härder (Universität Kaiserslautern). Bewertungskriterien waren die wissenschaftliche Qualität und die Bedeutung für das Gebiet.

Die Preisverleihung findet zum Abschluss der BTW'2003 statt. Die beiden Preisträger halten jeweils einen Kurzvortrag über ihre Dissertation. Schriftliche Kurzfassungen sind in diesem Tagungsband enthalten. Die Dissertationspreise gehen in diesem Jahr an

Reinhard Braumandl (Universität Passau) für die Dissertation
"Quality of Service and Optimization in Data Integration Systems"

und

Jens Lechtenbörger (Universität Münster) für die Dissertation
"Data Warehouse Schema Design".

Inhaltsverzeichnis

Eingeladene Beiträge

Web Services:
Distributed Applications without Limits
- An Outline -

Frank Leymann

IBM Software Group
Schoenaicherstr. 220
71032 Böblingen
Germany
LEY1@de.ibm.com

Abstract: Web services technology is all about distributed computing. There is no fundamentally new basic concept behind this and related technologies. What is really new is the reach of Web services and its ubiquitous support by literally all major vendors. Most likely, heterogeneity will at the end no longer be an obstruction for distributed applications. This will have impact on application architectures, middleware, as well as the way in which people will think about computing and businesses use computing resources. We sketch these impacts as well as some exemplary research work to be done to actually build the outline environment.

1 Introduction

Since the advent of Web services about two years ago most software vendors have embraced this technology and support it in their products. The spectrum of products supporting Web services reach from database systems over application servers over standard applications to office suites; corresponding support in tools is already available too. And Web services became an integral aspect of modern system architectures (e.g. [13]).

In a nutshell, a Web service is a virtual component that can be accessed via multiple formats and protocols. Such a component can be located anywhere in the network, e.g. on a machine on a different continent or within a thread in the same operating system process. Consequently, the environment for Web services is heterogeneous and distributed from the outset. Furthermore, Web services support a service-oriented architecture in which requestors can discover Web services and dynamically bind to them. But the primary focus of Web service technology is communication between Web services themselves, i.e. requestors are again Web services. Thus, to make the corresponding heterogeneous, distributed, and dynamic discovery-based environment work in practice, interoperability is key and standards are a must. A whole stack of standards has already been proposed (e.g. WSDL [11], SOAP [5], UDDI [3], and WS-Security [23]) and others will follow (see for example the roadmaps in [10] and [21]). Based on these standards a set of interoperability profiles will be published that describe artefacts from collections

of Web services standards and its recommended collective usage to ensure interoperability across platforms and languages (e.g. [1]). We describe the overall Web service environment and underlying basic concepts in section 2.

Grid technology [15] is about to evolve towards a "virtualisation layer" for hosting Web services ([16], [42]). Corresponding environments are under implementation, for example for Java [39]. This will enable what has been called recently "utility computing" or "on demand computing" [20]. Section 3 sketches this development.

Applications in this environment will consist of two parts, namely collections of individual and autonomic Web services (i.e. components) and aggregation specifications defined as business processes [12]. This will make the two-level programming model (e.g. [44], [29]) pervasive and will even allow involving human beings in applications. The corresponding application structure is outlined in section 4.

Finally, Web services also need to be aggregated in a less structured manner: Corresponding aggregation models for Web services appear (e.g. [8], [31], [41]) that allow building unstructured collections of Web services. Section 5 sketches the basics.

We conclude in chapter 6 and present the draft of a high-level middleware stack that supports the execution of this kind of applications.

2 Virtual Components

Web service technology makes functions available independent of many aspects of the proper implementation of the Web Service: A requestor has no need to know the programming model chosen to implement a Web service, i.e. whether the Web service is implemented in procedural or object oriented manner, for example. The programming language used to implement a Web service is completely irrelevant for a requestor. It doesn't matter whether the Web service is based on functions of a monolithic application system or whether it is build as a component, and if it is a component what the underlying component model is (e.g. J2EE, .NET). Any specific formats and protocols assumed by the Web service for direct communication is irrelevant for a requestor, i.e. it is hidden whether the implementation of the Web service expects ASCII files or Java objects, or whether it is invoked via a local call, an RPC or via a message queue, for example.

The concept of a WSDL *port type* is used to define what functions a Web service provides, i.e. a port type specifies the interface of a Web service. Different WSDL *bindings* can be used to specify how these functions can be accessed via different formats and protocols, e.g. via SOAP over JMS, or via Java objects via method call. And a WSDL *port* defines an actual endpoint where these functions can be accessed according to a certain format and protocol, e.g. a queue name, or a class name and JNDI name. In this sense, a Web service is a *virtual component* that can be implemented in many different ways, e.g. by real components or by any other piece of executable code (see Figure 1). Especially, a Web service is not at all coupled with any kind of Web technology; because of this we will often simply use the term *service* instead of the Web service and we will use both terms interchangeably.

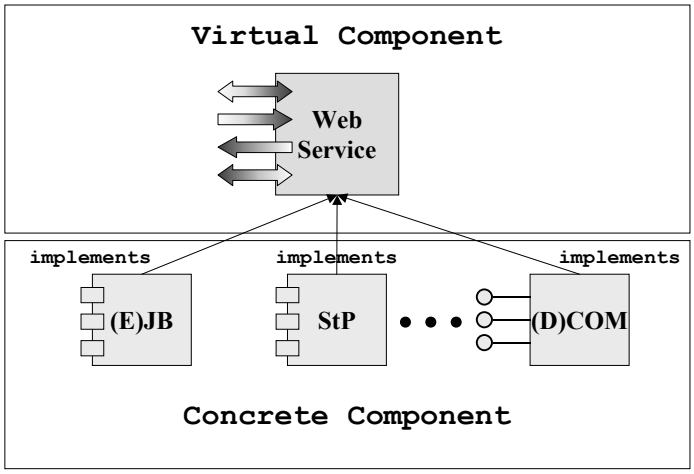


Figure 1 - Web Service as Virtual Component

2.1 Invocation

A user of a service should not be aware of the concrete implementation model chosen to realize the service: Whether the service is implemented as an EJB or a stored procedure or something else should be hidden as far as possible from the user. Thus, the user should be given a consistent "programming model" when dealing with services of different kinds. For this purpose, the environment of the user should provide features to deal with services of any kind in a manner specific to the environment and that appears seamless to the user.

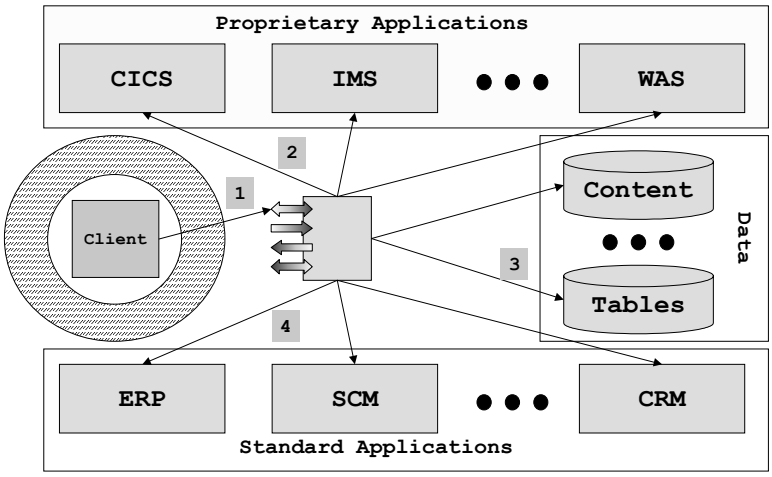


Figure 2 - Accessing Web Services

For example, a J2EE programmer should deal with Web services in a J2EE "style". In [26] we present a J2EE building block called WSIF (Web Service Invocation Framework – represented by the annular area in Figure 2) that exactly facilitates the latter; other environments may provide similar building blocks. In Figure 2, the client accesses a Web service (❶) in the programming model of its hosting environment (e.g. in Java based on WSIF); it doesn't even know that such different executables like a program in a TP monitor (❷), a table via an SQL statement (❸), or an ERP system (❹) may actually implement the Web service.

2.2 Lifecycle

A service can be statefull or stateless. For our discussion it is not important whether state is introduced via persistent instances or via session-like interactions. It is more important for our discussion whether or not the fact that a service is statefull or not is hidden from or visible to its clients: This has impact on the client programming model, i.e. whether a client has to explicitly manage the lifecycle of a service or not. When services are dynamically discovered, having to distinguish between statefull and stateless services causes complexity. Today, as a matter of fact, different application areas follow one approach or the other: In an OGSA Grid environment [42] statefull services are explicitly dealt with, while a BPEL business process environment [12] implicitly manages the statefullness of a service on behalf of a client.

At the level of details sufficient for us, OGSA uses an explicit factory-based approach to deal with the lifecycle of a Web service: A client uses a factory to create "an instance" of a particular kind of service. The client can then explicitly manage the destruction of such an instance, or it can be left to the Grid environment. In the latter case, a client registers its interest in the instance for a particular period of time (which can be extended). When no client is any longer interested in a given instance it can be destructed.

BPEL facilitates the implicit management of the lifecycle of an instance of a service via correlation identifiers embedded in messages: Application data exchanged with a service is assumed to carry enough information to identify a particular instance of a service. The state of a service is described via a process specification in BPEL. Depending on the actual state a service is in an incoming message results either in the automatic creation of an instance of a service, or the message is automatically routed to the appropriate existing instance. Finally, instances are automatically destructed when they reach their "final state".

2.3 Policies

Services need to describe their capabilities and requirements to their environment and potential users. A collection of capabilities and requirements is referred to as a *policy* [24]. A policy may express such diverse characteristics as transactionality, security, response time, pricing, etc. For example, a policy of a service may specify that all interactions must be invoked under transaction protection, that incoming messages have to be encrypted, that outgoing messages will be signed, that responses may only be accepted

within 5 seconds, and that certain operations are subject to a fee to be paid via credit card by the invoker.

Since policies might get quite complex they should be reusable. For this purpose, a policy can be specified as a separate document. Such a document can be associated with (constituents of) a Web service via an *attachment* [25]. Basically, an attachment consists of both, a policy and a subject the policy applies to ("resource"). Such subjects include port types, operations, messages, and also endpoints, i.e. individual ports or Web services, respectively. Attachments can be specified as follows (see Figure 3):

- Policies can be referenced out of the WSDL definitions of subjects. This method is suited to attach policies at the time when Web service resources are specified.
- Web services resources that are already deployed can be associated with policies by simply pointing to these resources and to the policies to be applied. Pointing to resources can be done based on *domain expressions* that describe the subjects and that have to be resolved in order to find the resources characterized by the policies. This method is especially suited to attach policies to existing resources.
- Finally, a policy can be registered itself in UDDI (as tModels). It can be associated with a UDDI business service (as key in a category bag).

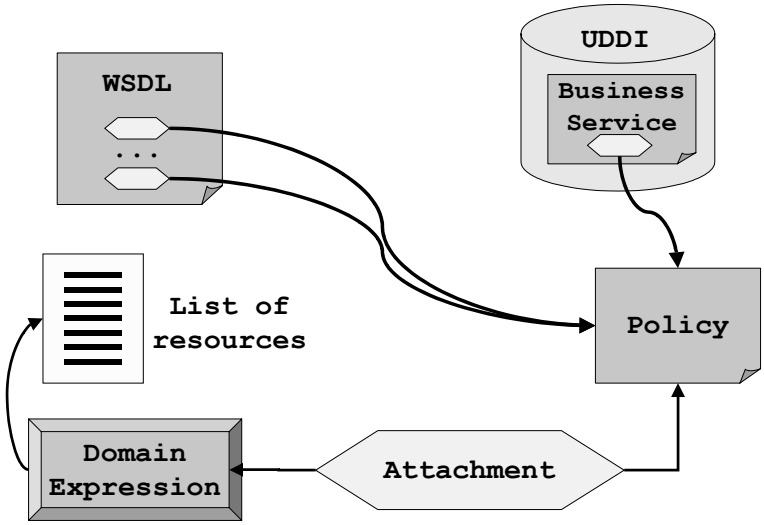


Figure 3 - Attaching Policies to Services

Service level agreements [34] can make use of policies and policy attachments. They do specify characteristics bilaterally agreed to in advance between the provider of a Web service and particular users. A service level agreement specifies aspects like committed

quality of services (like availability and average response time), payment methods for calling on a Web service, fees to be paid when service levels are not met etc.

2.4 Service Bus

Web service technology enables a new kind of architecture for composing applications referred to as *service oriented architecture* (SOA – see [7]). In SOA, services are registered in a service directory (e.g. in UDDI). Requestors find services they are interested in by enquiring service directories. The information they retrieve from a directory suffices to bind to a service and use it (see Figure 4).

When a service provider publishes a service in a service directory he specifies technical information about the service as well as business relevant information. Technical information about a service includes its interfaces, supported bindings, and endpoint information (e.g. the corresponding WSDL definitions). Business relevant information about a service falls into two categories: One category contains information about the suitability of a service from a functional perspective; the other category contains information about the suitability of a service from an operational perspective. The first category helps to understand whether a service is instrumental in achieving a business goal, e.g. buying a certain kind of sheet metal that is available within a certain period of time at a given price. Information provided are semantic descriptions about the kind of service facilitated by each of its interfaces, information about the service provider itself etc. The second category helps to understand whether a service satisfies the business policies of the requestor, e.g. all data are exchanged in an encrypted manner and are deleted once the trade is settled, messages are exchanged via reliable protocols, and payment is can be done once a month collectively for all orders. Information provided in this category includes payment methods, charging models, quality of services supported. The policy mechanism is expected to be used to describe this kind of information. Finally, all this information should be understandable by large communities, both, people as well as programs; it is expected that ontologies will play a major role in this area [14].

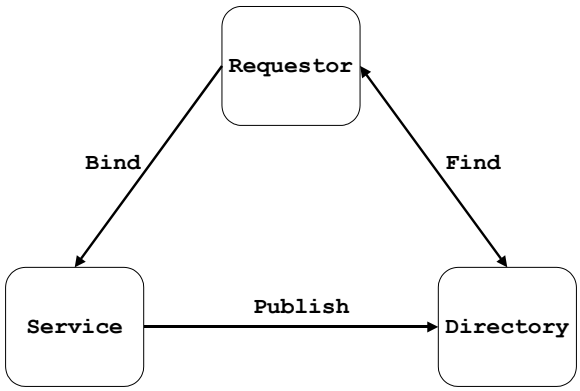


Figure 4- Service Oriented Architecture

Underlying SOA, there are really two distinguishing features: First, a requestor finds suitable services mainly based on queries in business terms (in contrast to technical terms). Second, the infrastructure hides as many technicalities as possible from a requestor. For example, a requestor specifies that he wants to analyze a gene based on a particular algorithm, and that he is wants to exchange all of the corresponding data encrypted. The infrastructure should find a service provider that matches the requestor's criteria and handle the corresponding request automatically on behalf of the requestor.

In Figure 5, this infrastructure is called *service bus*. The service bus receives the request and peals off the declarative description of the service required (❶). The description contains both, the business goals as well as the business policies of the requestor, and this description is used to derive the set of matching services offered by various service providers SP^j (❷). From a requestor's perspective, all qualified services are equivalent; i.e. the set of qualified services represent the *virtual service* (❸) described by the requestor by his request. If more than one service has qualified the service bus will decide on one of them (❹); this decision will be based on overall environmental properties like actual workload at the service provider side, average response time etc (e.g. measured or based on service level agreements with the service providers). Finally (❺), the service bus will bind to the service selected, pass the request message proper to it, and deliver the response to the requestor. Note that during step ❺ the invocation component sketched in section 2.1 is involved.

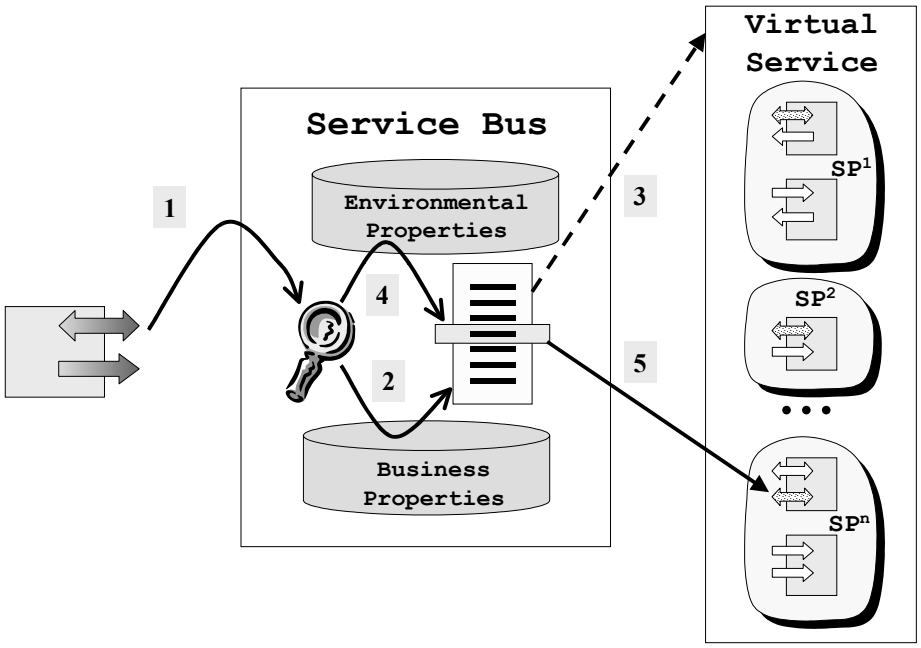


Figure 5 - Service Bus for Virtualizing Services

2.5 A Clarification

It should be clear until now that the sometimes-heard belief, Web service technology is all about SOAP, is erroneous. As shown above, Web Service technology is about SOA, a certain architectural style, which is far more than just SOAP: SOAP is primarily one particular wire-format used to exchange data as well as a set of conventions about how to appropriately process SOAP messages. The acronyms are close, but the goals are at different scale.

2.6 Sample Work to Be Done

Today, the component model underlying Web services is relatively simple. To allow more complex usage patterns ([19], [40]) work must be undertaken to define a more complex component model for Web services. For example, what mechanisms for component aggregation are required or desirable (see also section 5)? What are their advantages or disadvantages in a Web service world?

According to SOA Web services can be dynamically discovered and used. Today, the difference between statefull and stateless Web services is visible. Can and should the environment hide the difference and allow for a single client programming model? What is the impact of this programming model on the service bus?

Policies play a key role in the discovery of Web services. Often, policies are added to a Web service in an incremental manner. What are efficient algorithms to combine multiple policies into a single policy that describes a service or a request? A service as well as a request is decorated by a policy; how is matchmaking of policies done efficiently?

3 Virtual Operational Environments

The service bus introduced above virtualizes services: As long as a service qualifies under a request the service bus has the liberty to target the request to it. In doing so, the service bus can optimise the execution of a single request having the optimal exploitation of the overall environment in mind. It will use algorithms and mechanisms from scheduling, workload management etc that apply to the heterogeneous and distributed environment of Web services.

3.1 Grid Services

Middleware for scientific computing with similar goals has already been developed in the Grid computing area [15]. It thus seems only natural to bring the area of Grid computing and Web services together: [16] outlines an architecture for such a combined environment called Open Grid Services Architecture (OGSA). The most fundamental aspects of the special kind of Web services, called *Grid Services* that are hosted in such a combined environment are under specification (see [42]).

In order to become a Grid services, a Web service has to support a set of pre-defined interfaces and has to comply with some conventions. The interfaces to be supported facilitate the discovery, creation, and lifetime management of services; they further facilitate a notification mechanism to especially enable the manageability of services. The conventions deal primarily with naming services. Based on these interfaces and conventions a standard semantics for interacting with a Grid service is defined: How services are created, how their lifetime is determined, how to invoke functions of a service etc.

It is expected that many different environments, especially application server environments like J2EE [22] or .NET [2] will evolve to support Grid services. This would mean that the application server might provide a special container hosting these services or that existing containers are modified to support the semantics of these services (*Grid service container*). [39] describes the design of such a container based on both, native Java as well as on J2EE.

Such a (new or modified) container specifies the interface defining the interactions between the container and an implementation of a Grid service such that the implemented service appears to a requestor as a Grid service. As of today (year end 2002), this interface is not standardized; a corresponding standard would allow creating Grid services that are portable at least between homogeneous environments (e.g. J2EE compliant application servers). Nevertheless, requestors that use a Grid service based on the OGSA standard specified in [42] would be independent of the actual environment that hosts the Grid service used.

3.2 Grid Services Environment Stack

Based on [38], Figure 6 depicts the stack building the overall environment for applications of Grid services. At the bottom, it shows a Grid service container based on an environment like an application server; the container provides the functions discussed before. But the overall environment might consist of many different Grid service containers that are hosted on different autonomous and heterogeneous application servers. Thus, clustering capabilities are needed to "federate" the different Grid service containers resulting in a virtual environment for scalability and resource sharing. Also, such a virtual environment has to support distributed and heterogeneous problem determination and logging, the association of policies with Grid services as a base for request scheduling etc. The corresponding functions are referred to a meta-operating system services.

Often, collections of Grid services are needed to perform more complex functions that are not offered by individual services (see also section 5). Capabilities for managing such collections of services as well as making them jointly accessible are shown as a separate building block referred to as domain services. For example, domain services allow that individual instances of a particular Grid service type may join or leave a collection. Domain services also include functions for provisioning such collections to individual requestors.

At the top layer functions are shown that represent various autonomic services of the Grid: For example, Grid-wide workload management that enable a broad range of

mechanisms for scheduling requests in the Grid reaching from simple round-robin schedulers to policy-based meta-schedulers in hierarchical Grid topologies (see [4], [37]) enhancing overall availability and scalability within the Grid. Also, functions enabling utility computing (see next section) are at this layer.

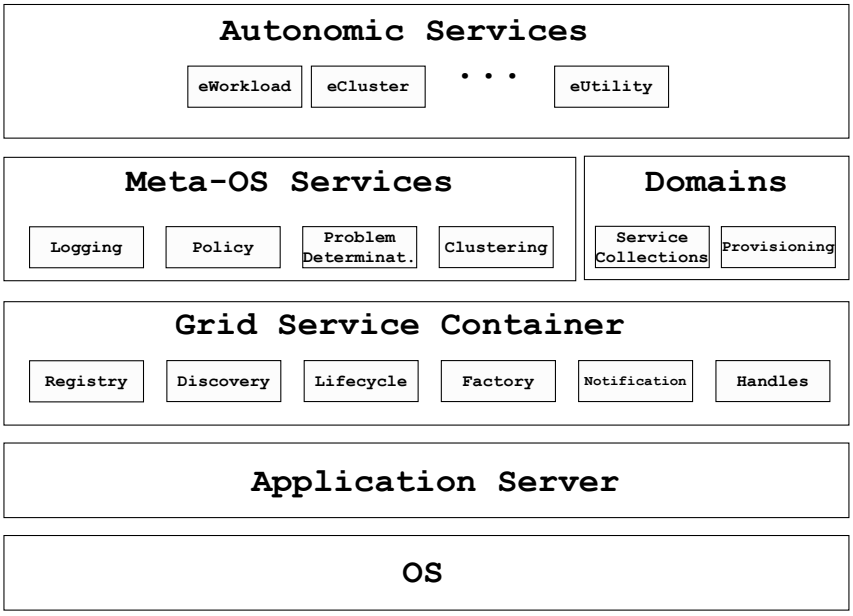


Figure 6 - The Grid Services Stack

3.3. On Demand Computing

Finally, such an environment will enable a new computing model called *on-demand computing* [20]. In a nutshell, this term refers to the ubiquitous availability of compute resources whenever needed and wherever needed. This bares the potential to turn computing into a public utility like water, power, gas, and telephone connections – which is why this model is also referred to as *utility computing*.

An important step on this path is represented by the concept of a hosted e-utility. A hosted *e-utility* is a collection of application-related services (both, hardware as well as all required software) that is made available by a service provider to a requestor on demand based on particular service level agreements for a certain fee. For example, a requestor wants to analyze new genomic data and needs for this purpose a set of certain algorithms, large amount of temporary storage, a set of servers to provide the corresponding compute power, as well a high-bandwidth connections to the Internet for access to public genomic data. A service provider can provide all of this as a collection of Grid services.

For this purpose, the service provider will make use of the Grid services stack sketched in the section before. For example, the required collection of services will be managed by the collection services. The collection will be assembled based on business rules and business processes depending on the ordered quality of services and negotiated service level agreements; for this purpose, the eUtility service of the autonomic layer can exploit workflow technology. Provisioning services are used to reserve the necessary resources to meet the service level agreement for the time period ordered. Note the relation between hosted e-utilities and application service providers (ASPs).

3.4 Sample Work to Be Done

It is obvious that there is a lot of work to be done to establish Grid computing and further on-demand computing as a broadly accepted model in practice. The spectrum of work reaches from low level technical work like specifying the agreed upon interfaces between the Grid service container and Grid service implementations such that these implementations become portable, over theoretical work on meta-schedulers, to business-related work on payment models, for example.

Finally, Web services and Grid services will further have to converge: It has to be investigated which properties currently specified as characteristics of Grid services do make sense in the more broader context of Web services, and which properties do only make sense in the more specific context of on-demand computing – if there are any such properties at all.

Note: We do not distinguish between Grid services and Web services in what follows and will often simply talk about services.

4 Application Structure

Services are either fine grained or coarse grained. From a requestor's perspective, a fine grained service achieves a business goal based on a single interaction, while a coarse grained service typically requires multiple interactions to achieve a business goal. Because a single interaction with a fine grained service suffices, a fine grained service typically does not reveal any of its inner structure, i.e. it is opaque hiding its implementation details. In contrast to this, a coarse grained service does reveal implementation details, especially the set of interactions required as well as their order, i.e. it is transparent making some of its inner structure visible to a requestor. The implementation details revealed by a coarse grained service describe its potential message exchange with the outside world, i.e. business rules that specify in which order and under which conditions which messages are sent to or expected from the requestor and perhaps other third party Web services. These details are important because it allows a requestor to determine whether he can interact with a particular service at all, for example.

4.1 Two-Level Programming Paradigm

In a Web services world actual messages are sent to ports via their corresponding operations. Thus, at the type level a potential message exchange can be specified by defining

the potential order in which operations of port types are used and under which conditions. As depicted in Figure 7 this is the same as specifying a business process or a workflow, respectively, the activities of which are realized by operations of port types (see [30]). Especially, a coarse grained service appears to be composed of the corresponding services, and consequently coarse grained services are also referred to as *composite services*. Vice versa, fine grained services are also referred to as *elemental services*.

In [12], a language called Business Process Execution Language for Web Services (BPEL for short) has been defined to specify how to compose a service from other services based on business process models (see [33] for a quick overview on BPEL). First, BPEL requires the specification of all of the port types a composite service offers to the outside world and in turn all port types from the outside world, which it expects to use. Second, it requires specifying the potential ordering in which operations of these port types may be used or have to be used, respectively, and this ordering can be specified dependent on business rules. I.e. a composite service is specified by sets of port types and a business process model exploiting operation of these port types.

This introduces the paradigm of two-level programming [44] to Web services: Programming in the small for implementing the elemental services used by a composite service, and programming in the large for specifying the composite service itself. Programming in the small, i.e. the implementation of elemental services, is done based on usual programming languages (e.g. Java, C#), and based on known component technologies and application server environments (e.g. J2EE, .NET). The corresponding components are hosted and rendered by the environment as Web services, i.e. the elemental services. Programming in the large is done based on a business process language (e.g. BPEL) hosted and run by a workflow system (see [29]). The corresponding business process is rendered again as a Web service resulting in a composite service.

In a nutshell, the set of port types offered by a composite service to the outside world represents the interface of this service. This notion of a service as an aggregate goes beyond WSDL, but offering just a single port type corresponds to the known notion of a service in today's WSDL. In section 5 we discuss other aggregation models for Web service.

BPEL can also be seen as a language for implementing a service based on other services. In this case, the Web service to be implemented is a composite service that offers a single port type to the outside world. If the services used to implement the composite service are publicly available the composite service is even portable to other environments that support BPEL, i.e. it will be able to be executed without any further actions; otherwise, the services used must be made available via appropriate deployment (see next section).

4.2 Reuse

The two-level programming paradigm introduces reuse at both levels: At the component level, i.e. elemental service level, and at the business process model level, i.e. composite service level. In practice, a vast number of isolated component functionalities does al-

ready exist in an enterprise, e.g. in form of purchased standard applications or home grown special applications. Typically, it is the knowledge of how to integrate these component functionalities into a business process that solves a (new) business problem. As a consequence, to become an artefact of reusability a business process model has to have the ability to be easily linked to the component functionalities available at an individual enterprise; a business process model with this property is sometimes called a *solution template* – or solution for short [32].

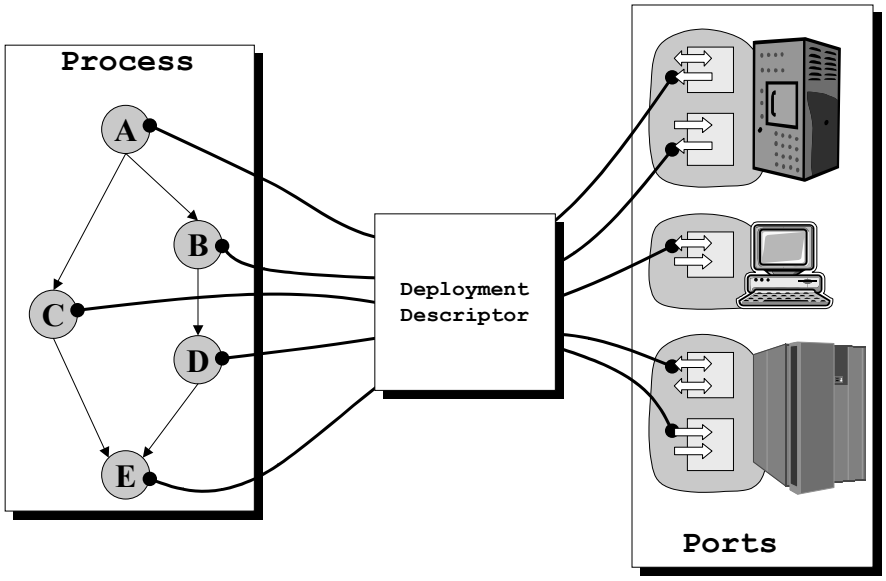


Figure 7 – Two-Level Application Structure

Linking a business process model to components is done in a step called *deployment* (see [22] for the original concept in J2EE, or [31] for a specialized concept called "locators"). During deployment for each port type referred to within the business process model it must be specified how it is bound to a corresponding port when an instance of the business process model is executed and makes use of an operation of a certain port type. *Binding* a port type to a port can be static or dynamic. *Static* binding assigns a fixed port to a port type. *Dynamic* binding assigns a mechanism to a port type that defines how a corresponding port is derived when needed at runtime. For example, one mechanism can be to assign a UDDI query to a port type that is to be evaluated at run time to determine a matching port. Another mechanism can be to expect a reference to the actual port to be used as a field in an incoming message (e.g. via service references and partner assignments in BPEL). The collection of deployment specifications associated with a business process model is called its *deployment descriptor*. Thus, a deployment descriptor links a solution template (i.e. a business process model) to existing ports, i.e. its turns a solution template into an executable solution (or application, respectively – see Figure 7). Those

ports might be both, elemental as well other composite services; the latter shows that the resulting programming model is recursive.

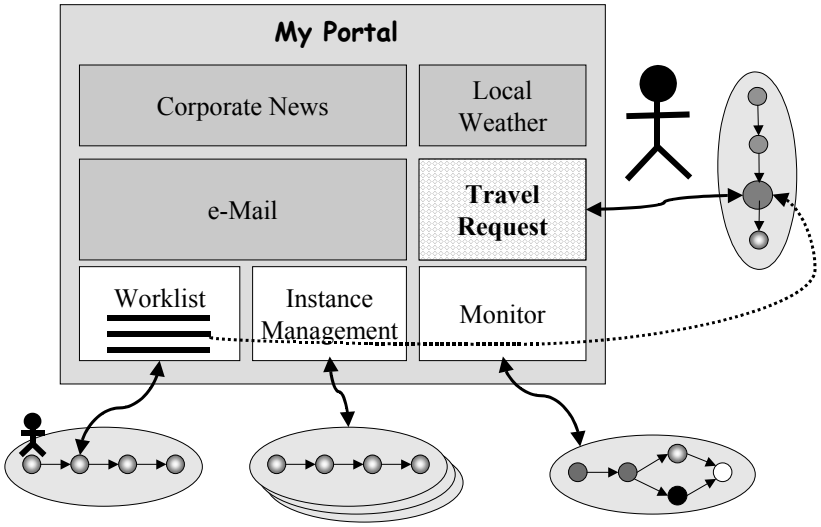


Figure 8 - Involving People in Business Processes

4.3 Involving People

Business processes may involve human beings [30]: A long running business process may be monitored by human beings interested in the actual state of the business process. Instances of business processes may be managed by human beings, e.g. an instance might be suspended and resumed later on. And a business process may involve people directly by creating work requests for certain people; these work requests are bundled into worklists for each person involved. A worklist may be perceived as a launch pad for tools supporting people in performing work requests: By selecting an item from a worklist the user initiates that the environment invokes the associated tool.

Typically, such an involvement of human beings in business processes is done today via portal technology. As shown in Figure 8, a portal may contain portlets that show a worklist of the person logged-on to the portal, and functions for the management and monitoring of business process instances. The tool to be launched when the person selects a work request from his worklist can be a Web service interacting with the person and that produces a user interface rendered within the portal (see [27] for the specification about how Web services can interact with portals).

For example, a person starts a business process for arranging a business trip based on instance management functionality made available in his portal. The corresponding business process first creates a work request for providing input about the trip to be arranged. This work request appears on the worklist of the person. When selecting the work request from the worklist the user interface appears that allows to key in the re-

quired data. Once this is done, the person can monitor the progress of his travel request via the monitoring functionality made available in the portal.

4.4 Sample Work to Be Done

Considering user-facing actions in the business process based two-level programming paradigm as well as the corresponding middleware aspects is something that has to be done in more detail. For example, how is a series of interactions with one and the same end user (i.e. a "dialog") reflected best in a business process? How is this related to the model-view-controller paradigm (e.g. [36]) that is typically used in environments that are not workflow-based? What is the relation between workflow-based implementations of dialogs and other implementation techniques for end user interactions (e.g. Struts or Java Server Faces [35])?

One aspect of BPEL is to put constraints on the possible usage of operations of (collections of) port types; this specifies a certain kind of semantics for the corresponding port types. How does this contribute to shape "the semantic Web"? Another aspect of BPEL is that of an executable language: How can BPEL support Grid applications, i.e. what modifications or extensions of BPEL are needed? For example, how can a Grid scheduler exploit workflow functionality, especially based on BPEL?

The concept of a solution template is worth to be considered further: Not only complete business process models are "templates" for application functionality but also "appropriate" fragments of a business process model. What properties characterize reusable fragments? How can fragments be expanded to become complete solutions? How is the semantics of a fragment changed when it is expanded?

5 Aggregation

The model of building a composite service as introduced in section 4.1 is one example of an aggregation model for Web services. In this model aggregation is done at the port type level by specifying both, the port types offered as well as required by the aggregate. Furthermore, the aggregation is very much structured and constrained in its behaviour by the associated business process model, i.e. it is "choreography"-centric: It prescribes the potential order in which the operations of the aggregated port types are to be used. And it is "pro-active" by defining an execution model that actually drives the usage of the aggregated port types. On the other hand, it is non-recursive in the sense that defining new port types based on its aggregated port types is not its focus.

Other aggregation models for Web services are possible:

- Aggregation models at the port type level focussed on the recursive definition of new port types (section 5.1).
- Aggregation models at the instance level (i.e. port level or service level, respectively) focused on (statically or dynamically) collecting services of certain port types without any assumption about structural relations between the services (section 5.2).

- Aggregation models at the instance level focussed on reaching outcome agreement between services that cooperate in a not explicitly prescribed manner (section 5.3).

5.1 Global Models

The definition of a recursive aggregation model (called *global model*) for specifying collections of new port types is included in [31] (see Figure 9). This model defines the notion of a *service provider type* as a set port types (e.g. SP^a). The only structural relation between service provider types is that they make use of each other's services. The relation between service provider types and the aggregate itself is that the aggregate's interface is built from the service provider types' interfaces.

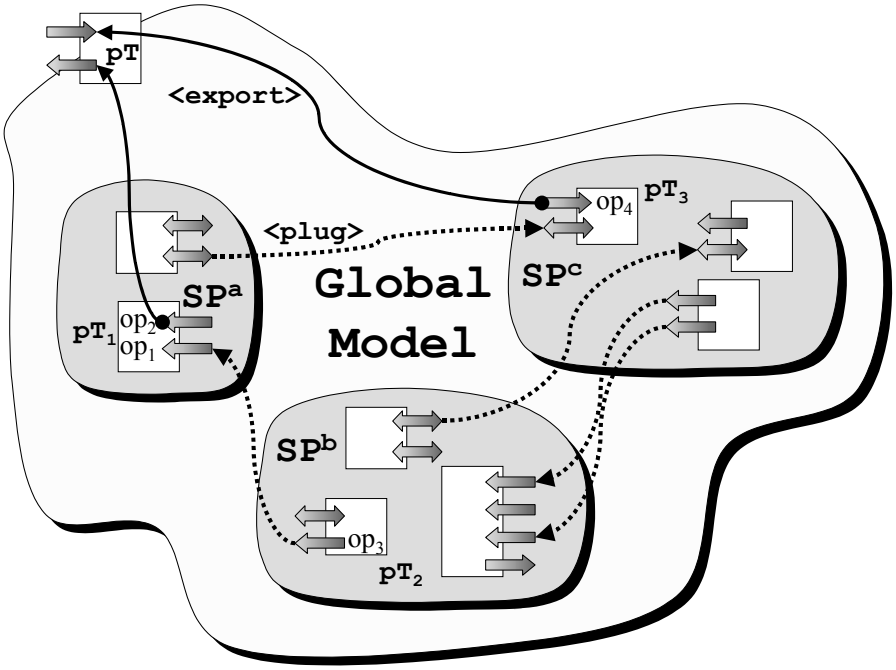


Figure 9 - Aggregation via Global Models

Operations of port types of different service provider types can be connected via a directed *plug link*. A plug link defines a client-server relationship between operations specifying who the initiator is and who the follower within an interaction is. For example, the out-operation op_3 of port type pT_2 of service provider SP^b is the source of a message send to the in-operation op_1 of port type pT_1 of service provider SP^a that consumes this message. It is not required that all operations are source or target of a plug link, i.e. a service provider might offer operations that are not used by other service providers of the aggregate. Furthermore, a plug link allows defining message transformations to handle

cases where the signatures of the linked operations do not match; for example, such a situation appears quite often in EAI environments.

The (new) interface of the aggregate is defined by *exporting* operations of constituent port types that are not used within a plug link. The semantics of exporting an operation is that the implementation of an operation from the interface of the aggregate is delegated to the operation of a port type of a service provider. For example, the in-operation of port type pT in Figure 9 is in fact the exported operation op_4 of port type pT_3 of service provider SP^c, i.e. if a requestor uses the in-operation of the aggregate the environment hosting the aggregate will forward the incoming message to op_4 of pT_3. The collection of service provider types, plug links and exports needed to define new port types make up a global model.

5.2 Service Domains

In some application scenarios, a requestor needs a collection of related services that he will use in a non-predefined manner. Properties beyond the signature level of a concrete service are irrelevant to a requestor, i.e. individual ports providing the same service are indistinguishable from a requestor's point of view. [41] specifies a complete environment for such aggregations; the corresponding aggregation model is referred to as *service domain*. For conciseness reasons, we will take the liberty here to use the same name but describe a variant of this aggregation model.

Basically, a service domain is a set of ports implementing a predefined set of port types. In general, for each particular port type associated with a service domain there is more than one port implementing this port type. A service domain aggregates these ports by providing for each of its port types a port that functions as a proxy for the set of ports implementing the same port type. When a requestor sends a message to this proxy the environment will select one implementing port and dispatch the message to it.

An extension of this base model introduces more dynamics: Providers can register and unregister ports with a service domain. Registration includes specification of the service levels (e.g. throughput, average response time) for the offered operations. Requestors are using services of a service domain based on formerly established service level agreements. Consequently, the environment will select implementing ports based on matching service levels and optimizing the utilization of the overall environment.

5.3 Coordination

Often, the final outcome of the usage of some services is dependent on the final outcome of the usage of some other services. As a result, an aggregation model is needed that allows dynamically creating temporary collections of services the joint outcome of their usage is determined once the period of usage of the services within the collection is over. The determination and dissemination of the joint outcome is based on a collection-specific set of protocols supported by the participating services, i.e. member of the collection.

Example 1: Consider a sealed-bid auction for divisible goods. Sellers inform the auctioneer about their goods to sell and buyers submit to the auctioneer the maximum price they are willingly to pay for a certain quantity of the good. Once the bidding period is over the auctioneer uses a clearing algorithm to determine the winners as well as the actual price each individual winner has to pay for his quantity. Finally, the auctioneer informs the seller about the winners as well as the corresponding prices and quantities, and he notifies winners and losers accordingly. Technically, the auctioneer, the seller, and the buyers are represented by appropriate Web services. When the seller offers his good he opens up a temporary collection of Web services that incrementally consists of his own service, the auctioneer's service, as well as the services of all bidders. The auctioneer, the bidders and the seller follow a certain protocol: First, the seller begins the trade by informing the auctioneer about the good to sell, then, the auctioneer is sending out request for bids, next, the bidders submit their bids to the auctioneer, and finally the seller notifies the seller, the winners, and the losers. After that, the temporary collection of services ceases to exist.

Example 2: Consider a distributed transaction that is managed via a two-phase-commit protocol ([18], [43]). An application begins a transaction with a transaction manager and issues manipulation requests to various resource managers. When the application has finished all of its manipulations it will closes the transaction by issuing a commit request to the transaction manager. The well-known two-phase commit protocol is then run amongst the players on the scene to determine the joint outcome of the transaction. Again, the application, the transaction manager, as well as the resources can be Web services. In the course of the transaction these services are dynamically aggregated into a temporary collection of services managed by a certain protocol set in terms of outcome agreement.

The corresponding abstraction at the Web service level – called Web Services Coordination (or WS-C for short) – has been specified in [8] (see [17] for a quick overview on WS-C): A *distributed activity* is a unit of computation that consists of a set of different services, and that requires to jointly agree on the outcome of the unit of computation between the constituting services. Agreement is reached base on *coordination protocols*. A coordination protocol is a collection of messages together with a prescription about how these messages are to be exchanged in order to reach agreement (e.g. the protocol between a bidder and an auctioneer, or a transaction manager and a resource manager). A coordination protocol is represented by a collection of port types, where each port type is the result of a logical grouping of messages appropriate for a class of participants in the protocol (e.g. the port types representing a bidder or an auctioneer, respectively). Coordination protocols are grouped into *coordination types*. A coordination type is a set of coordination protocols needed to reach agreement between the different kinds of services of a certain type of distributed activity (e.g. the protocols between seller and auctioneer, and bidder and auctioneer need to determine the outcome of a sealed-bid auction).

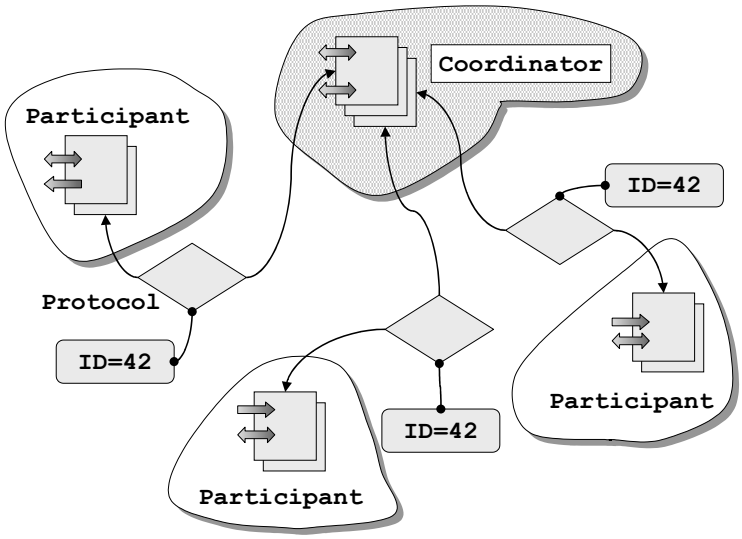


Figure 10 - Aggregation via Coordination

A *coordinator* provides services to create a distributed activity and to register for participation in a distributed activity (see Figure 10). An activity is created by specifying the coordination type used to agree on the outcome between the various participants. After creation, each activity has a unique *activity identity*. A service registers with an activity as *participant* by specifying the coordination protocol(s) it will honour. During registration, the participant and the coordinator exchange references to the ports that provide the operations allowing to mutually receiving the messages of the protocol(s). At the end of the activity the coordinator will communicate with each participant according to its registered protocol(s) to determine the agreed outcome of the activity.

WS-C specifies the framework sketched above but does not specify any coordination type. An accompanying specification called Web Services Transaction [9] (or WS-Tx for short) specifies coordination types for two major application areas: For "traditional" short-running distributed transactions, and for the extended transaction model for business processes specified in BPEL that allows to manage long-running units of work based on compensation actions (an extension of the model introduced in [28]).

5.4 Sample Work to Be Done

The aggregation models described in this section are not meant to be exhaustive. What other aggregation models are possible, and what are their application areas? How can the presented aggregation models be combined to become applicable in new areas (e.g. distributed activities across service domains)? Are there some sorts of "basic" aggregation models that can be used to describe other aggregation models?

6 Summary

In this paper we have demonstrated that Web services are the base for a new era of distributed computing. Web services are virtual components hiding from their users idiosyncrasies of the concrete (application server) technology chosen to implement the Web service. Especially, users can easily mix and match functions from heterogeneous environments into a single application if those functions are rendered as Web services. Based on a service-oriented architecture a user does not even have to care about a particular Web service he is communicating with because the underlying infrastructure, i.e. the service bus, will make an appropriate choice on behalf of the user. This choice is based on policies of both, the user and the Web services qualifying under the user's functional request, and the choice is also influenced by service level agreements and demand for an optimal utilization of the overall environment. We have shown that Grid computing technology and Web service technology are about to converge to provide these features and more, enabling utility computing and on-demand computing. Aggregations of Web services support a broad spectrum of requirements reaching from recursive component construction over advanced provisioning of groups of services to transaction management. Application construction and execution will be based on business process technology composing Web services into higher-level business functionality based on a two-level programming paradigm. Involvement of human beings in these applications is achieved by appropriate exploitation of portal technology. Figure 11 summarizes the corresponding middleware stack graphically.

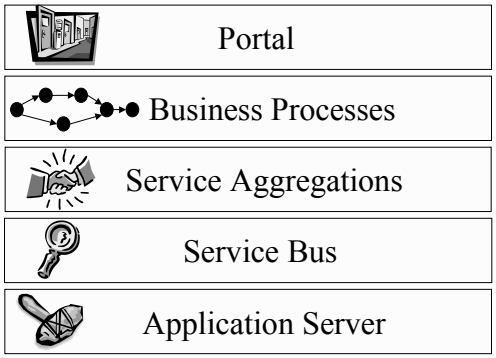


Figure 11 - Middleware Stack For Services-Oriented Applications

For the subject areas discussed in this paper we listed some selective research items. Many aspects that are of high importance for this kind of an environment like systems management, tooling for application construction and monitoring etc have not even been touched in this paper. Security aspects are utmost importance from a business perspective [6], but have been left out too. Payment methods, contracting etc appropriate in a Web service and on-demand environment are to be investigated, especially in situations in which services are recursively aggregated.

References

[1] K. Ballinger, D. Ehnebuske, M. Gudgin, M. Nottingham and P. Yendluri, Basic Profile Version 1.0, http://www.ws-i.org/Profiles/Basic/2002-10/BasicProfile-1.0-WGD.htm

[2] W. Beer, D. Birngruber, H. Mössböck and A. Wöß, Die .Net Technologie, dpunkt Verlag, 2003.

[3] T. Belwood et al, UDDI Version 3.0, http://uddi.org/pubs/uddi-v3.00-published-20020719.htm

[4] V. Berstis, Fundamentals of Grid computing, IBM Corporation (2002), http://www.redbooks.ibm.com/redpapers/pdfs/redp3613.pdf

[5] D. Box et al, SOAP 1.1, http://www.w3.org/TR/SOAP

[6] C. Boyens and O. Guenther, Trust is not enough: privacy and security in ASP and Web services environments, Proc. ADBIS 2002 - 6th East-European Conference on Advances in Databases and Information Systems (September 8-11, 2002, Bratislava, Slovakia).

[7] S. Burbeck, The Tao of e-business services, IBM Corporation, 2000, http://www-4.ibm.com/software/developer/library/ws-tao/index.html

[8] F. Cabrera, G. Copeland, T. Freund, J. Klein, D. Langworthy, D. Orchard and J. Shewchuk, Web Services Coordination, BEA Systems & IBM Coporation & Microsoft Corporation, 2002, http://www-106.ibm.com/developerworks/library/ws-coor/

[9] F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey and S. Thatte, Web Services Transactions, BEA Systems & IBM Coporation & Microsoft Corporation, 2002, http://www-106.ibm.com/developerworks/library/ws-transpec

[10] M. Champion, Ch. Ferris, E. Newcomer and D. Orchard, Web Services Architecture, http://www.w3.org/TR/ws-arch/

[11] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, WSDL 1.1, http://www.w3.org/TR/WSDL

[12] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte and S. Weerawarana, Business Process Execution Language For Web Services, BEA Systems & IBM Coporation & Microsoft Corporation, 2002, http://www-106.ibm.com/developerworks/library/ws-bpelwp

[13] B. Daum and U. Merten, System architecture with XML, Morgan Kaufmann Publishers, San Francisco, CA, 2003.

[14] D. Fensel, Ontologies: A silver bullet for knowledge management and electronic commerce, Springer, 2001.

[15] I. Foster and C. Kesselman, The Grid: Blueprint for a new computing infrastructure, Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[16] I. Foster, C. Kessleman, J.M. Nick and S. Tuecke, The physiology of the Grid – An open Grid services architecture for distributed systems integration, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002, http://www.globus.org/research/papers/ogsa.pdf

[17] T. Freund and T. Storey, Transactions in the world of Web services, http://www-106.ibm.com/developerworks/webservices/library/ws-wstx1

[18] J. Gray and A. Reuter, Transaction processing, Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[19] V. Gruhn and A. Thiel, Komponentenmodelle, Addison-Wesley, 2000.

[20] IBM, Living in an on demand world, IBM Corporation (2002), http://www-3.ibm.com/e-business/doc/content/feature/offers/whitepaper.pdf

[21] IBM and Microsoft, Security in a Web Services World: A Proposed Architecture and Roadmap, IBM Corporation and Microsoft Corporation (2002), msdn.microsoft.com/ws-security

[22] Java™ 2 Enterprise Edition, Java™ Platform Enterprise Edition Specification, Version 1.4, Sun Microsystems 2002.

[23] Ch. Kaler (ed.), Web Services Security, http:// www-106.ibm.com/developerworks/ web-services/library/ws-secure
[24] Ch. Kaler (ed.), Web Services Policy Framework, http://www-106.ibm.com/developerworks/library/ws-polfram
[25] Ch. Kaler (ed.), Web Services Policy Attachment, http://www-106.ibm.com/developerworks/library/ws-polatt
[26] D. König, M. Kloppmann, F. Leymann, G. Pfau and D. Roller, Web service invocation framework: A step towards virtualizing components, Proc. XMIDX'2003 (Berlin, Germany, February 17 – 18, 2003).
[27] A. Kropp, Ch. Leue and R. Thompson (editors), Web Services for Remote Portlets, Working draft 0.85 (OASIS, 26 November 2002), http://www.oasis-open.org/committees/wsrp
[28] F. Leymann, Supporting business transactions via partial backward recovery in workflow management systems, Proc. BTW'95 (Dresden, Germany, March 22-24, 1995), Springer, 1995.
[29] F. Leymann and D. Roller, Workflow based applications, IBM Systems Journal 36(1) (1997).
[30] F. Leymann and D. Roller, Production Workflow, Prentice Hall Inc., Upper Saddle River, New Jersey, 2000.
[31] F. Leymann, Web Services Flow Language, IBM Corporation (2001), http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
[32] F. Leymann, D. Roller and M.-T. Schmidt, Web services and business process management, IBM Systems Journal 41(2) (2002).
[33] F. Leymann and D. Roller, Business processes in a Web services world, IBM Corporation 2002, http://www-106.ibm.com/developerworks/library/ws-bpelwp/
[34] H. Ludwig, A. Keller, A. Dan, and R. King: A Service Level Agreement Language for Dynamic Electronic Services. Proceedings of WECWIS 2002, Newport Beach, CA, pp. 25 - 32, IEEE Computer Society, Los Alamitos, 2002.
[35] C.R. McClanahan (ed.), Java Server Faces, Sun Microsystems, Inc., 2002 - http://java.sun.com/j2ee/javaserverfaces/.
[36] S. Middendorf, R. Singer and J. Heid, Java, dpunkt Verlag, 2003.
[37] S. Mullender, Distributed systems, ACM Press, 1993.
[38] J. Nick, OGSA – Framework for Grid Service Evolution, Presentation at OGSA Early Adopters Workshop, Argonne National Lab, CA (May 29 – 31, 2002), http://www.globus.org/ogsa/events/JeffNickOGSAFramework.pdf
[39] T. Sandholm et al, Java OGSI Hosting Environment Design: A portable Grid service container framework, http://www.globus.org/ogsa/java/OGSIJavaContainer_2002-07-19.pdf
[40] D. Schmidt, M. Stal, H. Rohnert and F. Buschmann, Pattern-orientierte Software Architektur, dpunkt Verlag, 2002.
[41] Y.-S. Tan, B. Topol, V. Vellanki and J. Xing, Implementing service Grids with the service domain toolkit, IBM Corporation, 2002
[42] S. Tuecke et al, Grid service specification, http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-04_2002-10-04.pdf
[43] G. Weikum and G. Vossen, Transactional information systems, Academic Press, San Diego, CA, 2002.
[44] G. Wiederhold, P. Wegner, S. Ceri, Towards Megaprogramming: A paradigm for component-based programming, Comm. ACM 35(22) 1992, 89 – 99.

Data Integration: A Status Report

Alon Y. Halevy

University of Washington, Box 352350, Seattle WA 98195-2350, USA,
alon@cs.washington.edu

1 Introduction

Integration of data from multiple sources is one of the longest standing problems facing the database research community. In addition to being a problem in most large enterprises, research on this topic has been fueled by the promise of integrating data on the WWW. In the past few years, the community has made very significant progress on data integration, from the conceptual and algorithmic aspects, to the systems and commercial aspects. Below I briefly explain which areas we've made progress on, and what challenges lie ahead. This short paper is not meant to be a comprehensive survey of the state of the art in data integration, and represents only my personal perspective. I apologize in advance for the skewed list of references.

2 Recent Progress

Since the mid-90's, the database and AI communities have made significant progress in several aspects of data integration.

Schema mediation languages: a data integration system usually employs a *mediated schema*, which is purely a logical schema for the purpose of posing queries. In order for several autonomous data sources to interoperate, we need semantic mappings between the schemas of the different sources. The semantic mappings provide translations between the schemas of the sources and the mediated schema. Research on data integration has provided a set of rich and well understood schema mediation languages. The two commonly used formalisms are the *global-as-view* (GAV) approach used by [18, 22, 2], in which the mediated schema is defined as a set of views over the data sources; and the *local-as-view* (LAV) approach of [32, 12, 34], in which the contents of data sources are described as views over the mediated schema. The GLAV formalism [16] combines LAV and GAV. The semantics of the formalisms are defined in terms of *certain answers* to a query [1]. Surveys of these languages and their properties are given in [31, 24].

Query answering algorithms: associated with the mediation languages, significant progress was made on developing algorithms for answering queries in a data integration system. The specific problem of interest is to reformulate a

query posed over the mediated schema into a query that refers to the schemas of the data sources. In the case of the GAV formalism, answering queries reduces to view unfolding. In the LAV approach, answering queries translates to the problem of *answering queries using views* [24], for which several very efficient algorithms have been developed [38, 19].

Query optimization: the autonomy and heterogeneity of data sources present many new challenges to query optimization. In addition to the usual issues that arise in distributed query processing [37], in the data integration context data sources vary in their capabilities: some are simple web pages (or web-form interfaces to databases) and others are full-fledged relational query engines [22, 43, 15]. Handling *binding-patterns limitations*, i.e., the need to provide certain inputs to a data source in order to obtain answers has also received significant attention [40, 32, 14, 17, 11].

Query execution: the main line of work in the area of query execution for data integration systems was based on the observation that without statistics about the data sources, standard query optimization methods (i.e., cost-based optimization) do not apply. In answer to this challenge, several works considered *adaptive query processing*, where the system starts with some plan and adapts it as the execution proceeds [26, 28, 27, 3, 41, 6, 29, 42, 20]. The innovations in that work included the use of query operators that adapt as the data streams from the network, and different architectures for modifying query plans during execution.

Industry development: even three years ago, the term *data integration* in the commercial world referred explicitly to data warehousing. Practitioners and analysts were unwilling to believe any alternative architecture to data integration, mostly for two reasons: (1) lack of scalability and (2) inability to respect the autonomy of data sources. At the time, data warehouse products were already very well developed and accepted, and with the promise of *real-time* data warehousing, the need for other data integration architectures seemed diminished.

Today there is a data integration market. There is plenty of analyst coverage of different data integration products and architectures (often referred to as EII - Enterprise Information Integration). Companies such as IBM and BEA have announced initiatives leading to products in this realm, and a handful of startup companies such as Nimble Technology [10, 35] and Enosys [13] have delivered data integration products to customers, based on virtual integration architectures. One of the main issues facing these companies is the interaction between EII products and EAI (Enterprise Application Integration) products. It is reasonable to expect that in the near future we will start seeing integrated products from these two categories.

In summary, through years of research, development of prototypes and experimentation, we now know how to build the basic blocks of a data integration system. This is not to claim that the problems mentioned above are all solved, but a reasonable body of knowledge already exists, and now commercial efforts are starting to benefit from this body of knowledge.

3 Current Challenges

Despite the immense progress, several very significant challenges remain. I discuss a few of these below.

3.1 Generating Semantic Mappings

One of the major bottlenecks in building data integration applications (or any other application that requires multiple data sources to interoperate) is generating the semantic mappings between the data sources and the mediated schema. Currently, semantic mappings are written manually in a labor intensive and error-prone process. Tools that aid this process are crucial to the scalability of data sharing systems.

Unfortunately, complete automation of the generation of semantic mappings is unlikely to be possible. The crux of the problem is that writing a correct mapping requires an understanding of the underlying semantics of the schemas being matched. While the schema and data instances provide clues on their intended semantics, they do not fully capture the full semantics.

The problem of semi-automatically generating semantic mappings is an age-long problem in the database and AI communities (see [39] for a survey), and has recently received renewed attention [33, 8, 36, 9, 7]. Two principles are guiding the current work on this topic. First, a matching system needs to employ a collection of tools, each of which uses certain clues from the input to propose mappings. Second, a matching system should improve over time – every successful matching task provides experience to the system, and with the appropriate use of Machine Learning techniques, can be used to propose subsequent matches [8].

3.2 Peer-Data Management

Data integration systems are a significant step forward in data sharing systems, but still fall short of the grand vision of data sharing. In particular, consider the success of the World-Wide Web and that of P2P file sharing system. We still do not know how to create the analog of these two types of systems where the data being shared has rich semantics. The vision for sharing structured data has also recently been known as the *Semantic Web* [4]. Hence, an important challenge is to build data sharing architectures that enable such wide-scale authoring and querying.

One effort in this direction is known as *Peer-data management systems* (PDMS). PDMS offer a flexible architecture for sharing data, where instead of requiring the participants to create a central mediated schema, they can interoperate by specifying local semantic mappings, and queries can be answered by chaining mappings [21, 25, 30, 5]. As a result, sharing data is made easier, because peers can map their schema to the most convenient peer in the system rather than the mediated schema. In addition, they can query the system using their own schema, rather than using a mediated schema that may be foreign to them.

3.3 Crossing the Structure Chasm

The difficulties of creating large-scale data sharing systems are rooted in the profound differences between two types of data management: the unstructured world of text versus the structured world of data and knowledge bases [23]. In the structured world, authoring and querying data involve a conceptual effort of building and/or understanding a schema, while in the unstructured world, they merely involve writing text or entering keywords. The difficulties of sharing data in the structured world have been mentioned above, where as sharing is straightforward in the unstructured world. Clearly, one cannot expect management of data to be equally easy in both world – you need to *pay to play:* – you must put some effort into creating your data if you expect the benefits of complex querying offered by the tools in the structured world. However, there is a long way to go in making the interactions with data in the structured world much easier. In [23] this challenge is named *crossing the structure chasm.* This is a multi-decade project that will require substantial effort from the community.

4 Conclusion

Data management has traditionally focused on systems in which data is logically and physically centralized. The World-Wide Web, Ubiquitous Computing, Personal Data Management present challenges to data management where data needs to be created everywhere and by anyone, and accessed anytime and through various media. Data integration represents one point in the exciting journey from central data management to the vision of ubiquitous data. We have made significant progress over the last few years, and we must continue to focus towards the ultimate goal.

References

1. S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS*, pages 254–263, Seattle, WA, 1998.
2. S. Adali, K. Candan, Y. Papakonstantinou, and V. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of SIGMOD*, pages 137–148, Montreal, Canada, 1996.
3. R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proc. of SIGMOD*, 2000.
4. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.
5. P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing : A vision. In *ACM SIGMOD WebDB Workshop 2002*, 2002.
6. R. L. Cole. A decision theoretic cost model for dynamic plans. *IEEE Data Engineering Bulletin*, 23(2):34–41, 2000.
7. H.-H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proc. of VLDB*, 2002.
8. A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: a machine learning approach. In *Proc. of SIGMOD*, 2001.

9. A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proc. of the Int. WWW Conf.*, 2002.
10. D. Draper, A. Y. Halevy, and D. S. Weld. The nimble integration system. In *Proc. of SIGMOD*, 2001.
11. O. Duschka, M. Genesereth, and A. Levy. Recursive query plans for data integration. *Journal of Logic Programming, special issue on Logic Based Heterogeneous Information Systems*, 43(1):49–73, 2000.
12. O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proc. of PODS*, pages 109–116, Tucson, Arizona., 1997.
13. http://www.enosysmarkets.com, 2003.
14. D. Florescu, A. Levy, I. Manolesu, and D. Suciu. Query optimization in the presence of limited access patterns. In *Proc. of SIGMOD*, 1999.
15. D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3):59–74, September 1998.
16. M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proceedings of the National Conference on Artificial Intelligence*, 1999.
17. M. Friedman and D. Weld. Efficient execution of information gathering plans. In *Proceedings of the International Joint Conference on Artificial Intelligence, Nagoya, Japan*, pages 785–791, 1997.
18. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems*, 8(2):117–132, March 1997.
19. J. Goldstein and P.-A. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *Proc. of SIGMOD*, pages 331–342, 2001.
20. G. Graefe and R. Cole. Optimization of dynamic query evaluation plans. In *Proc. of SIGMOD*, Minneapolis, Minnesota, 1994.
21. S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? In *ACM SIGMOD WebDB Workshop 2001*, 2001.
22. L. Haas, D. Kossmann, E. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proc. of VLDB*, Athens, Greece, 1997.
23. A. Halevy, O. Etzioni, A. Doan, Z. Ives, J. Madhavan, L. McDowell, and I. Tatarinov. Crossing the structure chasm. In *To appear in the Proceedings of the First Biennian Conference on Innovative Data Systems Research (CIDR)*, 2003.
24. A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4), 2001.
25. A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *Proc. of ICDE*, 2003.
26. Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld. An adaptive query execution engine for data integration. In *Proc. of SIGMOD*, 1999.
27. Z. Ives, A. Halevy, and D. Weld. An xml query engine for network-bound data. *VLDB Journal, Special Issue on XML Query Processing*, 2003.
28. Z. Ives, A. Levy, D. Weld, D. Florescu, and M. Friedman. Adaptive query processing for internet applications. *Data Engineering Bulletin 23(3)*, 2000.
29. N. Kabra and D. J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proc. of SIGMOD*, pages 106–117, Seattle, WA, 1998.
30. P. Kalnis, W. Ng, B. Ooi, D. Papadias, and K. Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *Proc. of SIGMOD*, 2002.
31. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS*, pages 233–246, 2002.
32. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, Bombay, India, 1996.

33. J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.
34. I. Manolescu, D. Florescu, and D. Kossmann. Answering xml queries on heterogeneous data sources. In *Proc. of VLDB*, pages 241–250, 2001.
35. http://www.nimble.com, 2003.
36. N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2000.
37. M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, Upper Saddle River, New Jersey, 1999.
38. R. Pottinger and A. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB Journal*, 2001.
39. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.
40. A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proc. of PODS*, pages 105–112, San Jose, CA, 1995.
41. T. Urhan and M. J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Engineering Bulletin*, 23(2):27–33, 2000.
42. T. Urhan, M. J. Franklin, and L. Amsaleg. Cost based query scrambling for initial delays. In *Proc. of SIGMOD*, pages 130–141, Seattle, WA, 1998.
43. V. Vassalos and Y. Papakonstantinou. Describing and using the query capabilities of heterogeneous sources. In *Proc. of VLDB*, Athens, Greece, 1997.

Information System Architectures: From Art to Science

Peter C. Lockemann
Fakultät für Informatik
Universität Karlsruhe
Postfach 6980
76128 Karlsruhe
lockeman@ipd.uka.de

Abstract: The presentation claims that architectural design plays a crucial role in system development as a first step in a process that turns a requirements specification into a working software and hardware system. As such, architectural design should follow a rigorous methodology – a science – rather than intuition – an art. Our basic premise is that requirements in information systems follow a service philosophy, where services are characterized by their functionality and quality-of-service parameters. We develop a design hypothesis that takes the service characteristics into account in a stepwise fashion. We then validate the hypothesis for traditional database characteristics, demonstrate for novel requirements how these would affect architectures, and finally apply it to the current 4-tier server architectures.

1 Motivation

Information systems grow in the diversity of their application domains, number of users, and geographic distribution, but so does their complexity in terms of the number and functionality of components and the number of connections between these. An almost bewildering multitude of architectural patterns has appeared over the more recent past, that try to bring order into the evolving chaos. To name just a few of the buzzwords, take layered architectures, n-tier architectures, component architectures, middleware, vertical architectures, horizontal architectures, enterprise this-and-that. Nonetheless, it seems that these architectures have enough in common so that one suspects that they just look at similar phenomena from different perspectives, emphasize different aspects, or explore issues to different depths.

The premise of this paper is that architectural design plays a crucial role in system development. Unfortunately though, architectural system design does not seem to have too many friends. Typical excuses are that "top-down designs never work anyway because they ignore the technical possibilities and opportunities", that "even the cleanest architecture deteriorates over time due to the many additions and modifications on short notice", or that "architectures emphasize order over performance". We suspect that the real reason is the lack of a comprehensive, systematic and unifying approach to architectural design that makes the patterns in some sense comparable.

We claim that architectural design is the first step in a process that turns a requirements specification into a working software and hardware system and, hence,

could be seen as “programming-in-the-very-large”. Since it is an accepted doctrine that mistakes when caught in the early stages are much cheaper to correct than when discovered in the late stages, good architectural system design could be of enormous economical potential.

The purpose of this paper is to take a first step in the direction of a methodology for architectural design. Or in other words, we submit that architectural design should follow a methodology and not intuition, i.e., should be treated as a science and not as an art. In order not to become overly ambitious, and to stay within the confines of a conference paper, we will limit ourselves to information systems as the synthesis of data base and data communication systems, with more emphasis on the former.

2 Services

2.1 Services and resources

Since we claim that architectural design is the first step in a process that turns a requirements specification into a working software and hardware system, an essential ingredient of the design method is a uniform and rigorous requirements specification. Requirements is something imposed by an outside world. For information systems the outside world are the business processes in some real-world organization such as industry, government, education, financial institutions, for which they provide the informational support. Figure 1 illustrates the basic idea.

The counterpart of business processes in an information system are informational processes. Business processes proceed in a linear (as in Figure 1) or non-linear order of steps, and so do the informational processes. To meet its obligations, each step draws on a number of resources. *Resources* are infrastructural means that are not tied to any particular process or business but support a broad spectrum of these and can be shared, perhaps concurrently, by a large number of processes. In an information system the resources are informational in nature. Because of their central role, resources must be managed properly to achieve the desired system goals of economy, scale, capacity and timeliness. Therefore, access to each resource is through a *resource manager*. In the remainder we use the term *information systems* in the narrower sense of a collection of informational resources and their managers.

What qualifies as a resource depends on the scope of a process. For example, in decision processes the resources may be computational such as statistical packages, data warehouses or data mining algorithms. These may in turn draw on more generic resources such as database systems and data communication systems.

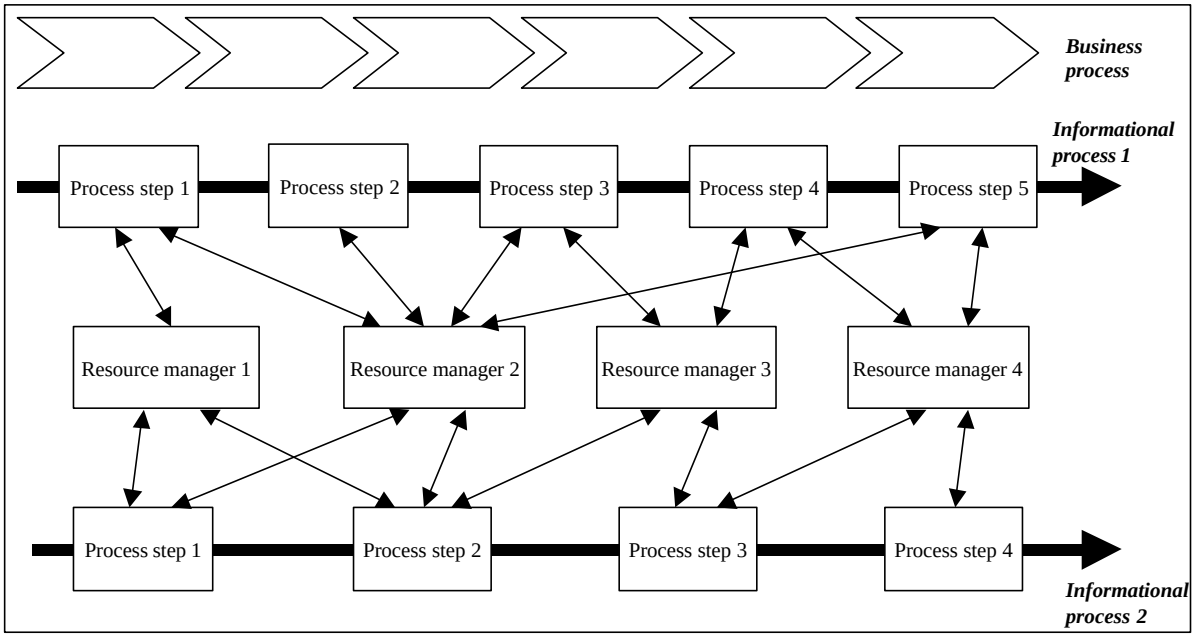


Figure 1
Business processes, informational processes and resources

What is of interest from an outside perspective is the kind of support a resource may provide. Abstractly speaking, a resource may be characterized by its *competence*. Competence manifests itself as the range of tasks that the resource manager is capable of performing. The range of tasks is referred to as a *service*. In this view, a resource manager is referred to as a *service provider* (or *server* for short) and each subsystem that makes use of a resource manager as a *service client* (or *client* for short).

2.2 Service characteristics

The relationship between a client and a server is governed by a service level agreement. In this agreement the server gives certain guarantees concerning the characteristics of the services it provides. From the viewpoint of the client the server has to meet certain *obligations* or *responsibilities*.

The responsibilities can be broadly classified into two categories. The first category is *service functionality* and covers the collection of functions available to a client and given by their syntactical interfaces (signatures) and their semantic effects. The semantic effects often reflect the interrelationships between the functions due to a shared state. Functionality is what a client basically is interested in.

The second category covers the *qualities of service*. These are non-functional properties that are nonetheless considered essential for the usefulness of a server to a client.

2.3 Service qualities

To make the discussion more targeted, we study what technical qualities of service we come to expect from an information system.

Ubiquity. In general, an information system includes a large – in the Internet even unbounded - number of service providers. Access to services should be unrestricted in time and space, that is, anytime between any places. Ubiquity of information services makes data communication an indispensable part of information systems.

Durability. Information services have not only to do with deriving new information from older information but also act as a kind of business memory. Access to older information in the form of stored data must remain possible at any time into an unlimited future, unless and until the data is explicitly overwritten. Durability of information makes database management a second indispensable ingredient of information systems.

Interpretability. In an information system, data is exchanged across both, space due to ubiquity and time due to durability. Data carries information, but it is not information by itself. To exchange information, the sender has to encode its information as data, and the receiver reconstructs the information by interpreting the data. Any exchange should ensure, to the extent possible, that the interpretations of sender and receiver agree, that is, that meaning is preserved in space and time. This requires some common conventions, e.g., a formal framework for interpretation. Because information systems and their environment usually are only loosely coupled, the formal framework can only reflect something like a best effort. Best-effort interpretability is often called (semantic) *consistency*.

Robustness. The service must remain reliable, i.e., guarantee its functionality and qualities to any client, under all circumstances, be they errors, disruptions, failures, incursions, interferences. Robustness must always be founded on a failure model. There may be different models for different causes. For example, a service function must reach a defined state in case of failure (*failure resilience*), service functions must only interact in predefined ways if they access the same resource (*conflict resilience*), and the effect of a function must not be lost once the function came to a successful end (*function persistency*).

Security. Services must remain trustworthy, that is, show no effects beyond the guaranteed functionality and qualities, and include only the predetermined clients, in the face of failures, errors or malicious attacks.

Performance. Services must be rendered with adequate technical performance at given cost. From a client's perspective the performance manifests itself as the response time. From a whole community of clients the performance is measured as throughput.

Scalability. Modern information systems are open systems in the number of both, clients and servers. Services must not deteriorate in functionality and qualities in the face of a continuous growth of service requests from clients or other servers.

3 Service hierarchies

3.1 Divide-and-conquer

Given a requirements specification in terms of service functionality and qualities on the one hand and a set of available basic, e.g., physical resources from which to construct them on the other hand, architectural design is about solving the complex task of bridging the gap between the two. The time-proven method for doing so is divide-and conquer which recursively derives from a given task a set of more limited tasks that can be combined to realize the original task. However, this is little more than an abstract principle that still leaves open the strategy that governs the decomposition.

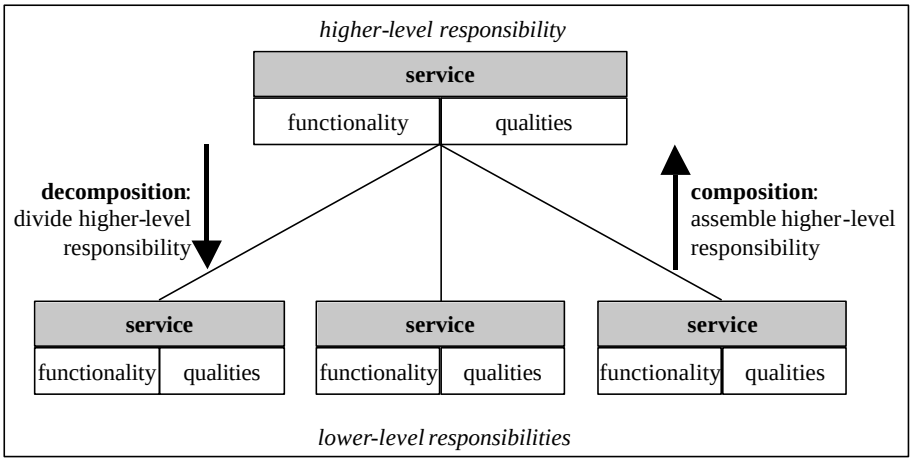


Figure 2 Divide-and-conquer for services

We look for a strategy that is well-suited to our service philosophy. Among the various strategies covered in [St02] the one to fit the service philosophy best is the *assignment of responsibilities*. In decomposing a larger task new smaller tasks are defined, that circumscribe narrower responsibilities within the original responsibility (Figure 2). If we follow Section 2.2, a responsibility no matter what its range is always defined in terms of a service functionality and a set of service qualities. Hence, the decomposition results in a hierarchy of responsibilities, i.e., services, starting from the

semantically richest though least detailed service at the root and progressing downwards to ever narrower but more detailed services. The inner nodes of the hierarchy can be interpreted as resource managers that act as both, service providers and service clients.

3.2 Design hypothesis

All we know at this point is that decomposition follows a strategy of dividing responsibilities for services. Services encompass functionality and a large number of quality-of-service (QoS) parameters. This opens up a large design space at each step. A design method deserves its name only if we impose a certain discipline that restricts the design space at each step.

The challenge now is to find a discipline that both, explains common existing architectural patterns, and systematically constructs new patterns if novel requirements arise. We claim that the service perspective has remained largely unexplored so that any discipline based on it is as yet little more than a *design hypothesis*.

Our method divides each step from one level to the next into three parts.

Functional decomposition.
This is the traditional approach. We consider service functionality as a primary criterion for decomposition. Since the original service requirements reflect the needs of the business world, the natural inclination is to use a pure top-down or *stepwise decomposition* strategy. At each design step a service functionality is given, and we must decide whether, and if so how, the functionality should be further broken up into a set of less powerful obligations and corresponding service functionalities to which some tasks can be delegated, and how these are to be combined to obtain the original functionality. However, the closer we come to the basic resources the more these will restrict our freedom of design. Consequently, at some point we may have to reverse the direction and use *stepwise composition* to construct a more powerful functionality from simpler functionalities.

Propagation of service qualities.
Consider two successive levels in the hierarchy and an assignment of QoS-parameters to the higher-level service, we now determine which service qualities should be taken care of by the services on the upper and lower levels. Three options exist for each quality. Under *exclusive control* the higher-level service takes sole responsibility, i.e., does not propagate the quality any further. Under *partial control* it shares the responsibility with some lower-level service, i.e., passes some QoS aspects along. Under *complete delegation* the higher-level service ignores the quality altogether and entirely passes it further down to a lower-level service. For partial control or complete delegation our hope is that the various qualities passed down are orthogonal and hence can be assigned to separate and largely independent resource managers.

Priority of service qualities.
Among the service qualities under exclusive or partial control, choose one as the primary quality and refine the decomposition. Our hope is that the remaining qualities exert no or only minor influences on this level, i.e., are orthogonal to the primary quality and thus can be taken care of separately.

Clearly, there are interdependencies between the three parts so that we should expect to iterate through them.

4 Testing the design hypothesis

4.1 Classical 5-layer architecture

Even though it is difficult to discern from the complex architecture of today's relational DBMS, most of them started out with an architecture that took as its reference the well-published 5-layer architecture of System R [As79, Ch81]. Up to these days the architecture is still the backbone of academic courses in database system implementation (see, e.g., [HR99]). As a first test we examine whether our design hypothesis could retroactively explain this (centralized) architecture.

4.1.1 Priority on performance

We assume that the DBMS offers all the service qualities of Section 2.3 safe ubiquity, and we ignore security for the time being. The service functionality is determined by the relational data model in its SQL appearance.

As noted in Section 2.3, durability is the raison d'être for DBMS. Durability is first of all a quality that must be guaranteed on the level of physical resources, by non-volatile storage. Let's assume that durability is delegated all the way down to this level. Even after decades durability is still served almost exclusively by magnetic disk storage. If we use processor speed as the yardstick, the overwhelming bottleneck, by six orders of magnitude, is access latency, which is composed of the movement of the mechanical access mechanism for reaching a cylinder and the rotational delay until the desired data block appears under the read/write head. Consequently, performance dwarfs all other service qualities in importance on the lowest level. Considering the size of the bottleneck and the fact that performance is also an issue for the clients, it seems to make sense to work from the hypothesis that performance is the highest-priority quality across the entire hierarchy to be constructed.

4.1.2 Playing off functionality versus performance

Since we ignore for the time being all service qualities except performance, our design hypothesis becomes somewhat simplified: There is a single top-priority quality, and because it pervades the entire hierarchy it is implemented by partial control. The challenge, then, is to find for each level a suitable benchmark against which to evaluate

performance. Such a benchmark is given by an access profile, that is a sequence of operations that reflects, e.g., average behavior or high-priority requests. We refer to such a benchmark as *data staging*.

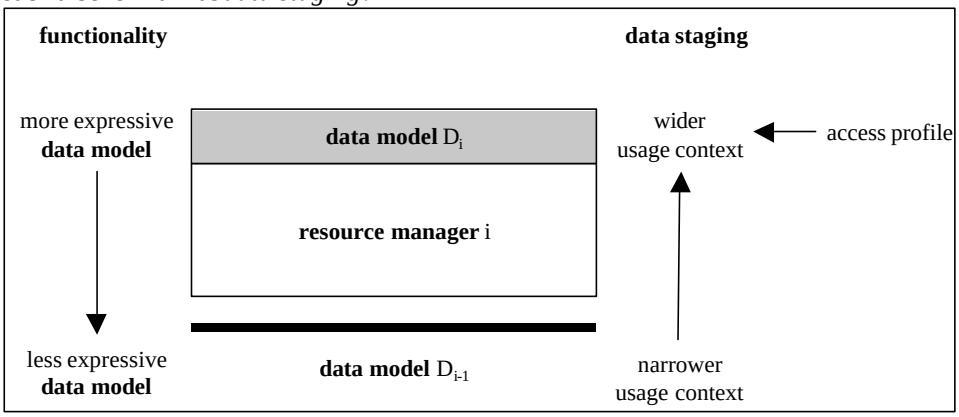


Figure 3 Balancing functionality and performance on a level

Consequently, our main objective on each level is determining a balance of functionality and data staging. As Figure 3 illustrates, the balancing takes account of a tandem of knowledge. On the way down we move from more to less expressive data models and at the same time from a wider context, i.e., more global knowledge of prospective data usage, to a narrower context with more localized knowledge of data usage. The higher we are in the hierarchy, the earlier can we predict the need for a data element. Design for performance, then, means to put the predictions to good use. Based on these abstractions we are indeed able to explain the classical architecture.

- We start with the root whose functionality is given by the relational model and SQL. The logical database structure in the form of relations is imposed by the clients. We also assume an access profile in terms of a history of operations on the logical database. We compress the access profile into an access density that expresses the probability of joint use of data elements within a given time interval. The topmost resource manager can now use the access density to rearrange the data elements into sets of jointly accessible elements. It then takes account of performance by translating queries against the relational database to those against the rearranged, internal database. The data model on this internal level could very well still be relational. But since we have to move to a less expressive data model, we leave only the structure relational but employ tuple operators rather than set operators. Consequently, the topmost resource manager also implements the relational operators by programs on sets of tuples.
- What is missing from the access density is the dynamics – which operations are applied to which data elements and in which order. Therefore, for the next lower level we compress the access profile into an access pattern that reflects the frequency and temporal distribution of the operations on data elements. There is a large number of so-called physical data structures tailored to different patterns –

take hash algorithms for associative access, list structures for sequential access, trees for combined associative and sequential access. The resource manager on this level accounts for performance by assigning suitable physical structures to the sets of the internal data model. The data model on the next lower level provides a library of physical data structures together with the operators for accessing them.

- It is not all clear how to continue from here on downwards because we have extracted all we could from the access profile. Hence we elect to change direction and start from the bottom. Given the storage devices we use physical file management as provided by operating systems. We choose a block-oriented file organization because it makes the least assumptions about subsequent use of the data and offers a homogeneous view on all devices. We use parameter settings to influence performance. The parameters concern, among others, file size and dynamic growth, block size, block placement, block addressing (virtual or physical). To lay the foundation for data staging we would like to control physical proximity: adjacent block numbering should be equivalent to minimal latency on sequential, or (in case of RAID) parallel access. The data model is defined by classical file management functions.
- The next upper level recognizes the fact that on the higher levels data staging is in terms of sets of records. It introduces its own version of sets, namely segments. These are defined on pages with a size equal to block size. Performance is controlled by the strategy that places pages in blocks. Particularly critical to performance is the assumption that record size is much lower than page size so that a page contains a fairly large number of records. Hence, under the best of circumstances a page transfer into main memory results in the transfer of a large number of jointly used records. Buffer management gives shared records a much better chance to survive in main memory. The data model on this level is terms of sets of pages and operators on these.
- This leaves just the gap to be closed between sets of records as they manifest themselves in the physical data structures, and sets of pages. Given a page, all records on the page can be accessed with main memory speed. Since each data structure reflects a particular pattern of record operations, we translate the pattern into a strategy for placing jointly used records on the same page (record clustering). The physical data resource manager places or retrieves records on or from pages, respectively.

Figure 4 summarizes the discussion.

4.1.3 Taking consistency into consideration

Data models represent generic functionalities, that is they are described by polymorphic type systems. Consequently, the managers in the service hierarchy of Section 4.1.2 deal with databases and queries generically. On the other hand, databases and queries against them must be monomorphic to be able to process them. Data models are instantiated to monomorphic type systems by specifying database schemas that are derived from the application semantics. Hence, consistency manifests itself in database schemas.

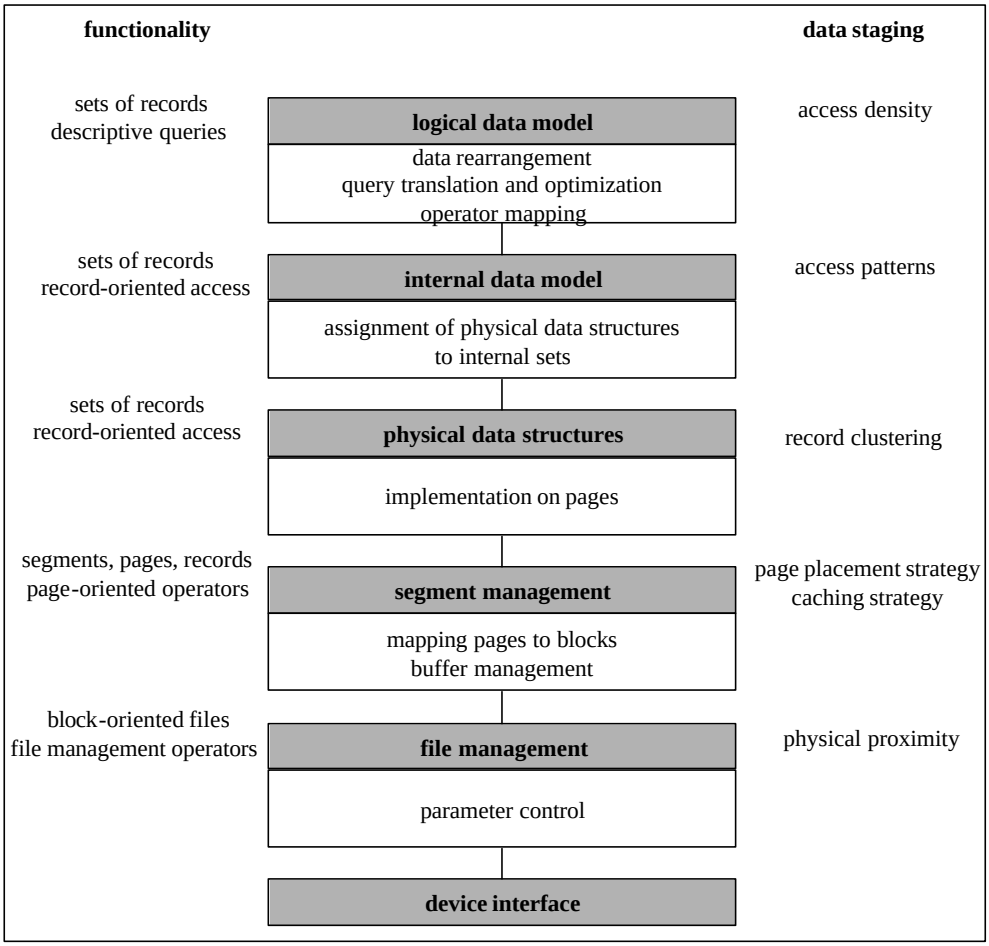


Figure 4 Reference service hierarchy for set/record-oriented database management systems

We conclude from the design process in Section 4.1.2 that only the upper two managers for logical and internal databases need a rich type system. Hence, consistency is the responsibility of just the upper two managers. Access density and patterns must be expressed in terms of the database schema to make sense.

Both managers use the schema to interpret the queries and to control the performance. Both access the schema but have no need to manage by themselves the functions for accepting, checking, storing and retrieving the schema. Nor does any of the lower managers appear to be a candidate to which to delegate this functionality. Consequently, we add a new service, meta data management, that is used by the two managers. As a service shared by two levels it seems to fall outside the hierarchy of Figure 4 (Figure 5). With the new service we may again associate service qualities such as (meta-)consistency or durability so that the design process should be repeated for the new branch.

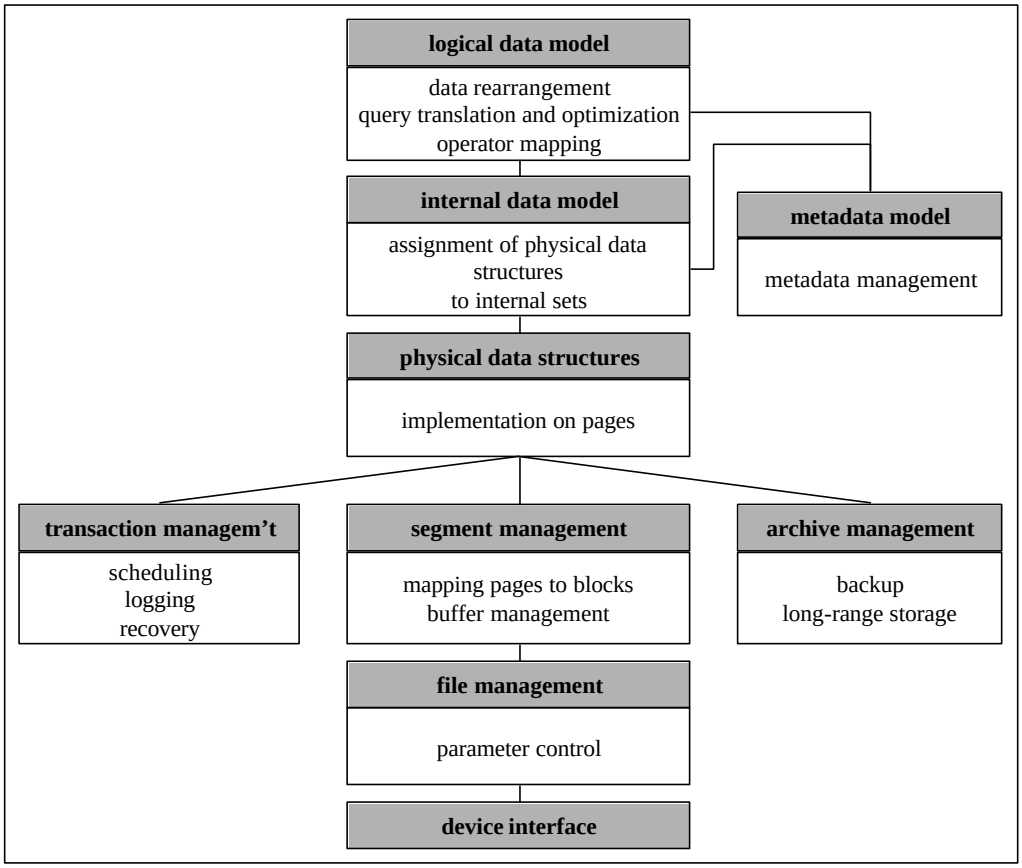


Figure 5 Augmented reference service hierarchy of Figure 4

4.1.4 Adding robustness

Consistency in terms of the database schema guarantees that the database reflects a possible state of the environment. Transactions can ensure that the database corresponds to the current state. Consequently, robustness with failure resilience, conflict resilience and function persistency is defined in terms of transactions as atomic processing units. With the exception of Weikum's multi-level transactions [We88] all reference architectures deal with transactional qualities (at least if they are ACID) on levels that are devoid of application semantics. In the design of Section 4.1.2 the proper level for transaction management would then be the segment level.

The segment level must now take care of two service qualities, performance and robustness. According to our design hypothesis we have to decide if the two are orthogonal. Intuitively one indeed considers them orthogonal because one would like to have performance even if there was no robustness, and robustness even if performance was not an issue. As a result one would split the segment level into two resource

managers, segment management proper and transaction management. Figure 5 shows the extension of the architecture. In fact, the two qualities are not entirely independent of one another so that the two managers should closely communicate with one another.

4.1.5 Adding durability

Peripheral storage may itself be subject to technical failures or external catastrophic events. Consequently, there is more to durability than what we considered in Section 4.1.2. The arguments in Section 4.1.4 can be repeated here: Durability is a generic property that ought to apply irrespective of application semantics, and it is orthogonal to performance and robustness (to the latter because of its much longer time horizon). Consequently, we add another resource manager, archive management, on the segment level (Figure 5).

4.2 Semistructured databases

Our design hypothesis held up quite well to explain the classical 5-layer architecture both in its core and its extensions. A somewhat harder test would be to try and apply the hypothesis to an area where there is less agreement as to the best reference architecture: semistructured databases or more specifically, XML databases.

4.2.1 Front-ends

For the sake of comparison with Section 4.1, let us assume that there is still consistency to be observed, that is, there is a database schema (either DTD or XML Schema). Under these circumstances we currently find two approaches. One imposes external factors, either technical such as interoperability between DBMS on the basis of XML as the data exchange format, or economical such as minimal cost of re-implementation. The result is some kind of XML front-end to a relational DBMS. The second approach builds a tailored, so-called native DBMS for XML.

In terms of our design hypothesis, one could explain the front-end as singling out the external factors as additional qualities of service that are kept under exclusive control. The XML data model significantly differs from the relational data model, though. The data structures are hierarchies rather than flat tuple sets, access is navigational by path expressions rather than set-algebraic, the nodes in the tree may have structural differences even if they satisfy the same type, and because of the document history nodes may include long texts or other media data. Consequently, the backside of the approach is that the front-end, if it deals with performance at all, does so on criteria that differ from the rest of the system.

4.2.2 Native systems

There is no reason to believe that performance plays a lesser role for (pure) XML databases. Therefore the design process of Section 4.1.2 based on seamless

performance should apply here as well, though we should expect that the differences in the data model have a significant effect.

- The root functionality is now given by XML, with the logical database structure in the form of trees and reading access by path expressions that identify subtrees. For writing access there is as yet no common standard. Some vendors prefer simple delete/write for modifications, or experiment with XSLT. Consequently, there is no clear way to separate access density and patterns – both take the form of navigated trees. Density is complicated by the fact that nodes have large structural variances as to number of attributes, cardinality of same-tagged successors and size of attached media data, and patterns allow various selection choices due to the structural variances. Indeed, native XML database products seem to collapse the two upper resource managers of the relational system into a single, fairly complex manager.
- The next lower resource manager is now something akin to the physical data structures. Basically one would expect three kinds of data structures: Subtrees of XML structures, index structures for navigating through XML structures, and media data of possibly large size.
- Each of these physical structures can then be optimized with regard to performance. Index structures come closest to the relational situation and may thus be realized on the segment level similar to Section 4.1.2. Subtrees may vary widely in size so that either large pages sizes must be chosen or subtrees may span a number of pages. Both require solutions that differ somewhat from that typical for relations, so that segment management becomes definitely more complicated. Media data would extend across many pages, moreover classical buffer caching would make little sense for them. They would, therefore, directly draw on the services of file management.

Figure 6 summarizes the discussion. Viewed superficially the architecture looks simpler than for the relational case. In fact, though, it is just less structured because it seems more difficult to decompose the services. In the end each resource manager in the architecture is more complicated than in the relational case. We note in passing that the problem of media data is also known as well for relational systems where large binary fields are used for the purpose.

5 Putting the design hypothesis to work

Section 4 seems to bear out the validity of our design hypothesis. But does it really? Or – so one might suspect – did we just formulate the hypothesis to fit the well-established architectures? Better proof would be to find constructive solutions to some novel situations.

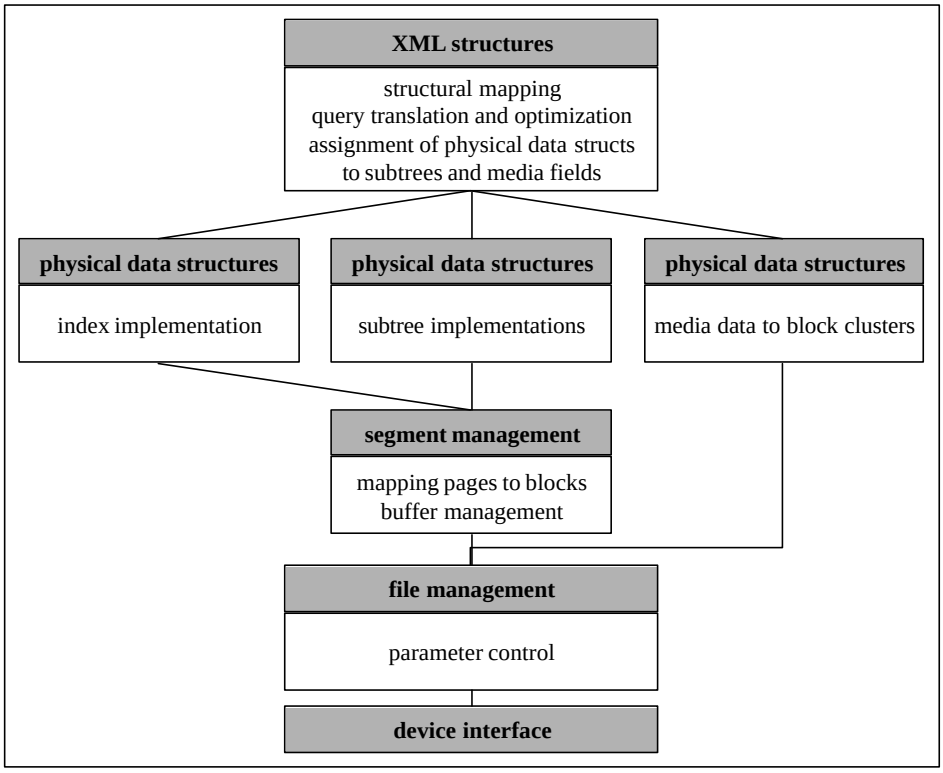


Figure 6 Service hierarchy for semistructured database systems

5.1 Hippocratic databases

In a recent paper [Ag02] Rakesh Agrawal et al. introduce the concept of *Hippocratic databases* for database systems that should enforce all political and societal rules to protect the privacy of data. The authors translate the requirement into the service quality of *purpose*: Any acquisition of data, their durability over a time span, and any queries on them have to serve a specific purpose. A purpose should be formulated in a precise fashion, all persons affected should agree, and there should be no access to the data without the purpose. The authors then go on to propose an architecture. The first impression is that the authors pretty much stick to the service hierarchy of Section 4.1 and just augment it by additional, albeit complex components. In other words, the authors do not question the prevalence of performance.

What would have happened if they had done so? If they had made purpose their prime quality? Let's try for a hypothetical answer (Figure 7). Suppose that we query the database in the usual way, except that the query is accompanied by its purpose. We expect that the result is those data whose purpose is compatible with the purpose of the

query. All other data remain invisible to the client, i.e., become visible neither by accident nor by intent.

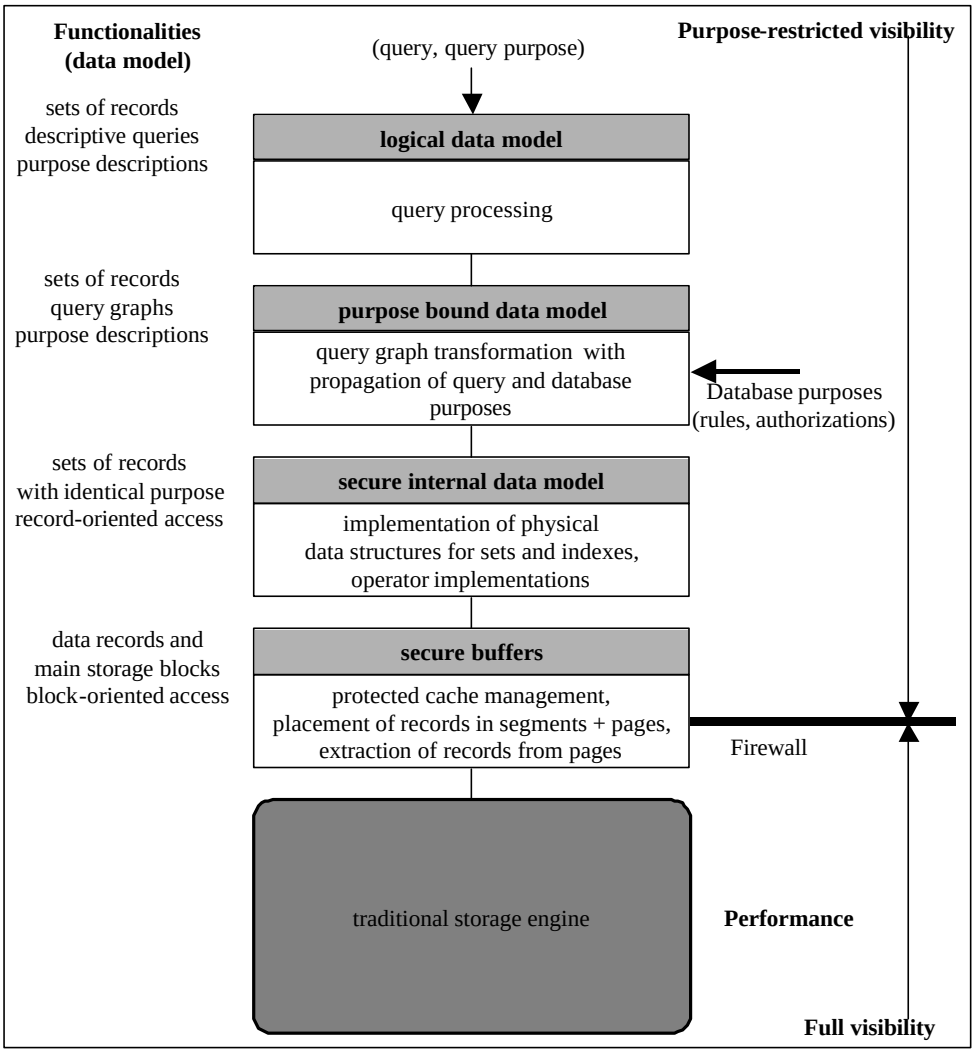


Figure 7 Hypothetical service hierarchy for Hippocratic databases

Notice that the purposes of the database are not open to inspection by regular clients. Consequently, they can only be seen from the second level on downwards. The uppermost level will confine itself to standard query processing into, e.g., a query graph together with information on the query purpose. This is input to the second level together with the rules, e.g., authorization rules, on database purposes. This allows to compute the visible nodes in the query graph. The third level sets up the physical data

structures for the visible data and computes their materializations. In other words, this level will separate the visible data from the invisible ones and reproduce the former in a separate main storage area. The next lower level is the most critical one because it must do the physical separation, for instance in a well-protected buffer area, and thus is something akin to an internal firewall. Only below can we proceed with the traditional storage engine from the level of segment management on downwards. And only from there on replaces performance purpose as the prime quality.

There is an apparent conclusion: Privacy protection in DBMS comes at the price of lower performance.

5.2 Incorruptible databases

While Hippocratic databases restrain access according to purpose no matter what the intent, *incorruptible databases* restrain changes no matter what the purpose. More precisely, the goal of incorruptible databases is to allow only those updates that satisfy certain rules, i.e., that make sense or seem plausible. Incorruptibility can be seen as a special kind of consistency. Consistency thus becomes the prevailing quality of service.

Plausibility can be tested in various ways. For example, one may employ statistical and data mining techniques to detect outliers, or one may formulate somehow restricted first-order logic formulae to check new input in the light of absolute constraints, the current database state or the previous history of database states. The second approach has a lot of similarity to deductive databases. We try to follow this approach.

Deductive constraint checking is known to be compute-bound rather than I/O-bound. Clearly then, performance is again the quality of priority. But in contrast to Section 4 where it was due to durability it is now due to consistency. Moreover, whereas in Section 4 we assume queries to be spontaneous or any of a large number of parameterized queries, consistency constraints are stable over a long period of time so that preprocessing makes sense. Further, a large number of constraints must be observed at any one time so that optimization techniques can employ additional techniques such as common subexpressions [He96]. Consequently, the service hierarchy splits at the top into a preprocessing ("compile") service and a query-and-update ("just-in-time") service. The later does the just-in-time rule processing employing the results of the precompiled checking modules. Deductive databases usually employ relational databases that have been augmented by special algebraic operators. Both precompilation and just-in-time optimization should generate queries that fetch into a special main memory cache those database portions to be checked, but that are formulated in such a way as to delegate the optimization with respect to data staging to traditional levels of a DBMS. Figure 8 summarizes the hypothetical solution.

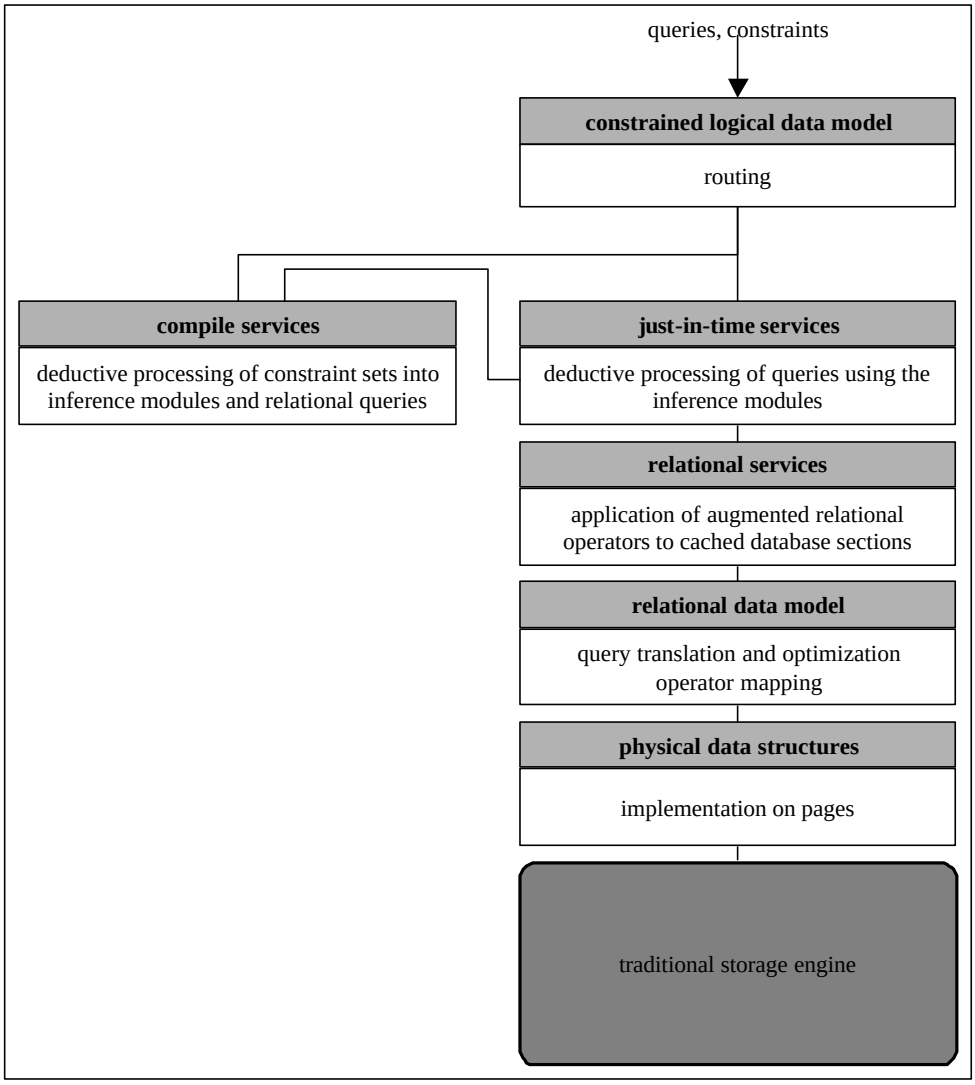


Figure 8 Hypothetical service hierarchy for incorruptible databases

6 Tight and loose coupling

6.1 Layered and component architectures

The reader may have noticed that what we have constructed so far were service hierarchies. These are not yet architectures but just the preliminary steps towards an architecture.

System architectures fall into two broad categories, *layered architectures* and *component architectures*. Two successive levels in a service hierarchy form successive layers if there is a close coupling between the two. If the coupling is loose, the levels form separate components.

Intuitively speaking, close coupling reflects a higher degree of interdependence, loose coupling a lesser degree. In view of the broad meaning of interdependence it will be difficult to find a single factor that determines the strength of coupling, and therefore to give a precise formulation. Instead we take a phenomenological approach.

- One conceivable criterion for loose coupling is that the two services either share no resources, or do so free of conflicts. An example is services that run on different computers or in different processes.
- One may consider as a case of loose coupling if two levels are separated by exclusive control or by complete delegation, that is, if they attend to different service qualities. Hence, the two share no responsibilities, and they can be expected to pursue independent strategies and thus to achieve a fair degree of autonomy.
- Even in case of partial control there may be a case for loose coupling if satisfying the quality is achieved by moving in different directions, indicating that different conditions come into play. Just remember the reversal of direction during the design of the classical 5-layer architecture in Section 4.1.2 due to the disappearance of application semantics.
- From software engineering it is well-known that the traffic density between two resource managers, e.g., the number of service calls or messages per time unit, determines the needed strength of the coupling. Higher densities require close coupling to avoid a decline in performance.
- Finally, a service may be of general value to different service hierarchies and thus be called from various clients. In this case close coupling would give an unfair advantage to just one client.

These criteria may occur in combination, and are not even completely orthogonal. Consequently, in a given situation there may still be a choice of whether to translate a hierarchy into a layered or a component architecture.

6.2 Service hierarchies revisited

One can easily demonstrate that with the four criteria above there is more than one architectural solution for the hierarchy of Section 4.1 (Figure 5). Fairly straightforward is the case for metadata and archive management. Both have their own resources, the top two levels of the hierarchy usually call on metadata only during certain start-up phases, and archive management kicks in only at larger intervals. Hence, both are natural components.

The boundary between physical data structures reflects a reversal of direction. Therefore one may opt for two components comprising the levels above and below, respectively. On the other hand, the two levels share the buffers located in the segment level. As a consequence, there is also a high traffic density between the two by placing and accessing records. Further traffic arises from resolving conflicts between the two. And indeed, during the long history of database systems both, complete layering and separation into two components, can be found, although the latter has mostly been confined to prototypes.

Similar considerations hold for transaction management. Transaction management has (almost) exclusive control over robustness, suggesting a component solution. However, usually transaction management carries some responsibility for performance, indicated by sharing the buffer with segment management. Traditionally, the performance argument won the day, and transaction management migrated into segment management. An exception is the QoS of conflict resilience which has lately even shown a tendency to move outside the DBMS.

Figure 9 illustrates the extreme of full componentization.

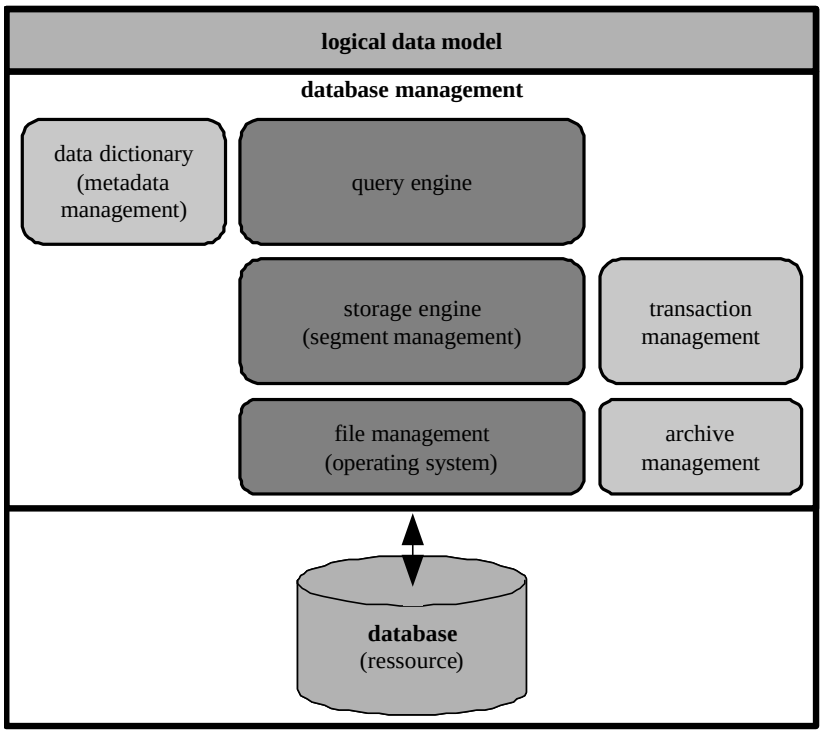


Figure 9 Component architecture for the service hierarchy of Figure 5

One may apply a similar design strategy to the other service hierarchies. For example, for the hierarchy of Section 4.2 (Figure 6) there is a somewhat stronger though still not overwhelming argument for providing separate storage engines. In Figure 7 the firewall mandates loose coupling, with a separate storage engine and perhaps part of the secure buffers level placed in a separate component. In Figure 8 the compile services qualify as a component, and the boundary between relational services and relational data model has many traits of loose coupling.

We conclude that from a programming-in-the-very-large standpoint, if given a service hierarchy, components determine a coarse-granular system architecture and layers the fine-granular architecture within a component.

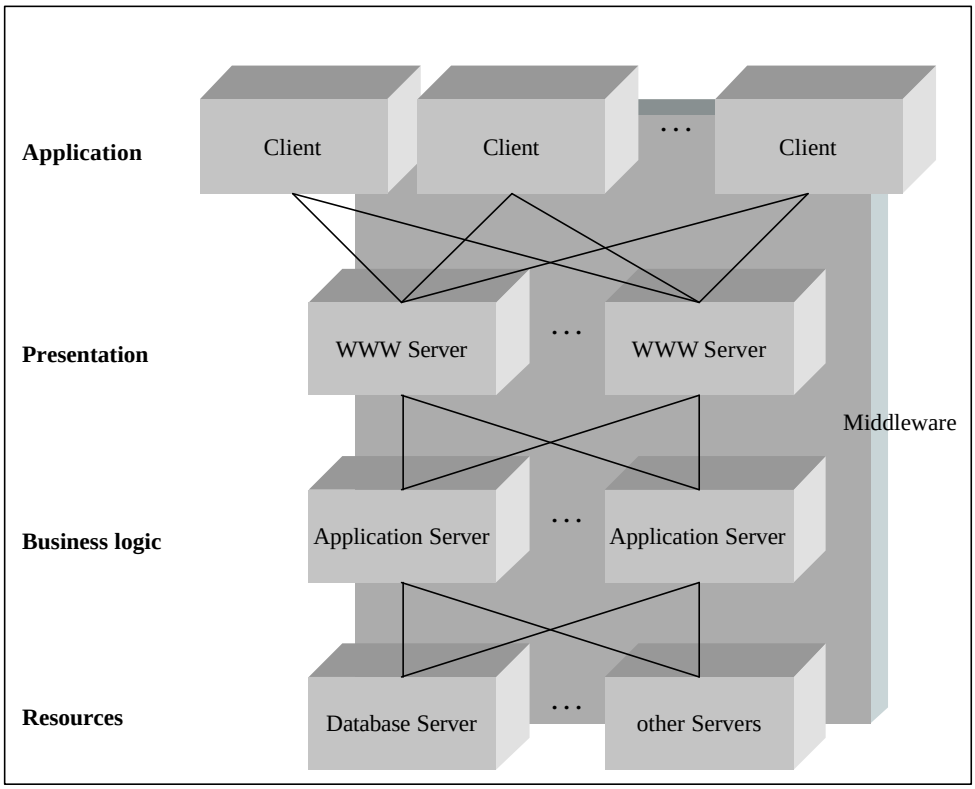


Figure 10 Four-tier architecture in a multidimensional framework

7 Coarse-granular component architectures

7.1 Tiered and multi-dimensional architectures

Information systems are more than just database systems. Indeed today, database systems are almost entirely hidden behind middleware and application servers. On the other hand, modern information systems are still service providers so that our design approach should carry over to these much coarser architectures.

As just mentioned, coarse architectures should be component architectures. The services may still be part of a service hierarchy. If this is the case we speak of a *tiered architecture*. A modern example is the 4-tier architecture with presentation clients, presentation servers, application servers and data managers (Figure 10).

The 4-tier architecture needs an infrastructure for communication and interoperability, commonly referred to as middleware. Figure 10 illustrates how the middleware as a component is called from each level in the service hierarchy so that it seems to pervade the entire hierarchy. One may visualize such an arrangement as the hierarchy forming one dimension and the common component (which in all likelihood is itself a service hierarchy) defining a second dimension. We refer to such an arrangement as a *multi-dimensional architecture*.

To illustrate the principle, consider as a very simple example Figure 11 [Ab03]. It shows a database server together with some application client. Both communicate across a transmission network with its own layered architecture (in our case the bottom four OSI/ISO levels). Space defines the third dimension.

7.2 Design hypothesis revisited

Take the architecture of Figure 10. Three of the four component types exhibit a relatively narrow and standardized functionality: client, Web server and database server. The specifics of an application – the business logic in the form of service functionality and QoS parameters – are concentrated in just one component, the application server.

Compared to Sections 4 through 6 the situation is now different: The component architecture is not of our own design but is being imposed – for good software engineering reasons. All we can hope for is that each component does its best to meet its QoS parameters. But as we all know, a set of local optima does not necessarily result in the global optimum. To approach the latter, we modify our design hypothesis as follows.

1. Assign service qualities to each component.
2. If the component can be influenced, develop it according to the design hypothesis of Section 3.2.
3. For each quality shared between adjacent components and arranged in order of priority, increase by technical means the strength of the coupling.

4. Re-evaluate and adjust the adjacent components in the light of the coupling technique.

In essence, because we can no longer start with a clean slate we employ a heuristic that first develops local optima and then, by iteration, tries to approach the global optimum.

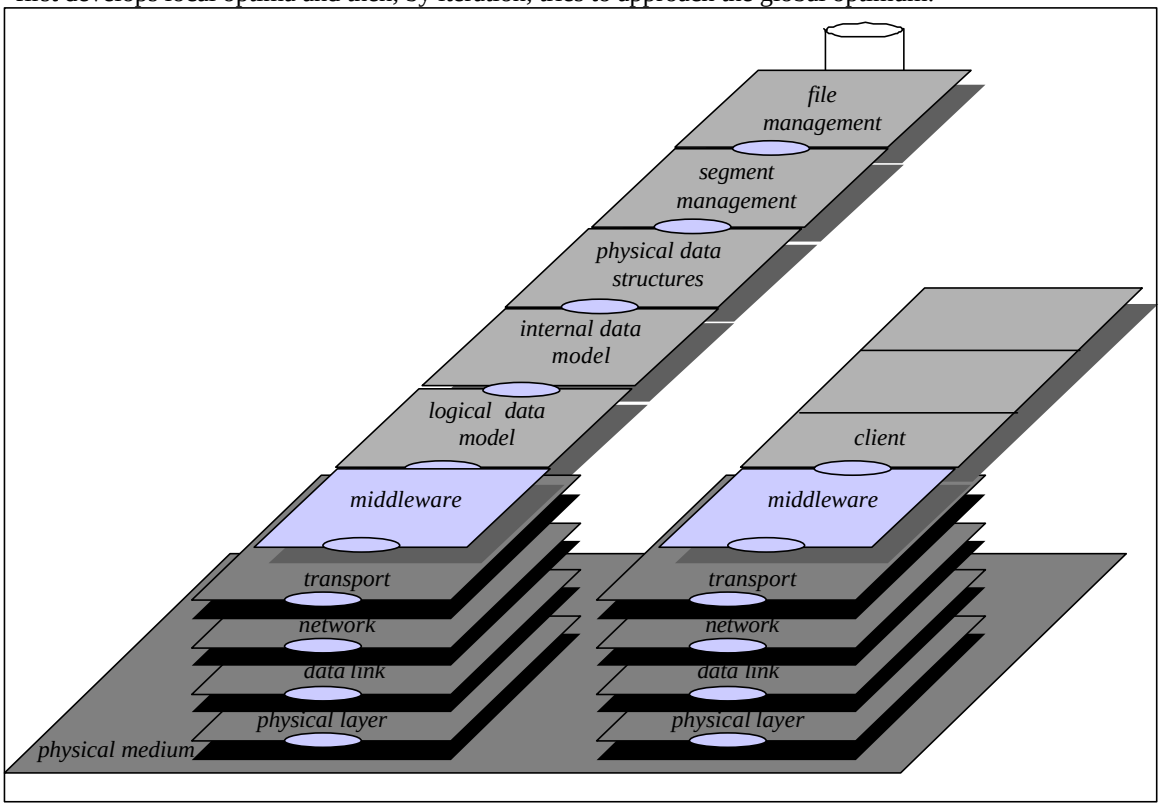


Figure 11 Multidimensional architecture linking a client to a database system via data communication

7.3 The design hypothesis at work

7.3.1 Priority on performance

Suppose that as in Section 4 performance is our highest-priority quality. Consequently, our design goal is to minimize the performance loss across the coupling between adjacent components. The basic principle should remain the same as in Section 4.1.2: Employ *data staging* to take the access profile into account. In the centralized architecture of Section 4.1.2 and with page size exceeding record size by a large factor a single buffer in the storage engine proved sufficient. For loose coupling, though, we will have to replicate and rearrange data as we move upwards in the hierarchy and consider progressively wider context, as we already did in some sense in Figure 7. In

other words, data staging via buffering, or as it is more generally called, by *caching* will be needed for each component coupling. Figure 12 reflects the principle.

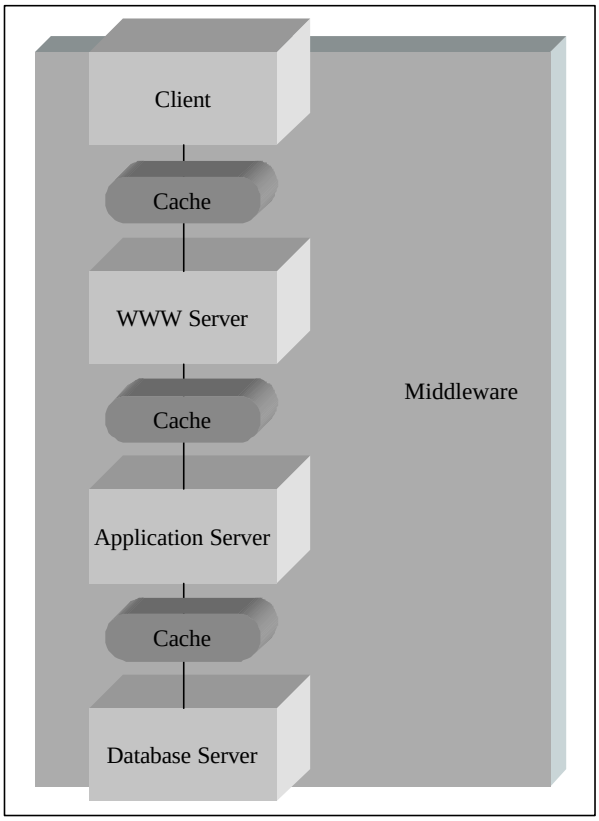


Figure 12 Performance control in a four-tier architecture

Translating the conceptual architecture into a physical architecture is by no means trivial, though. Several options exist for a cache.

- Separate component. The major advantage is that the autonomy of each component is maintained. No intrusion is necessary, each participating component remains as it is. On the other hand, all communication with the cache has to go through the middleware, adding considerable overhead and, hence, reducing the benefits of caching.
- Integration into the middleware. As before, the autonomy of each component is maintained. Data communication overhead is reduced because the buffer is part of the middleware. Because caching now becomes a generic middleware service one may encounter difficulties in trying to tailor it to the data staging needs of a specific connection.

- Integration into a participating component. We have to decide which component to choose and, hence, to modify for the purpose. If we employ the context argument then this should be the component in the hierarchy which has the best knowledge of the context.
- Migration. The cache is attached, i.e., physically coupled to the middleware or one component (edge-of-net or edge-of-server). The advantages of edge-of-server are the same as for a separate solution, but with lesser data communication overhead. Somewhat more coordination overhead arises between component and cache as compared to full integration.

These options give rise to a wide range of combinations and thus to a large optimization space, as pointed out by Mohan [Mo01]. Examples abound. Caches for the connection between application server and database server, so-called application data caches, can be edge-of-database-server if they materialize database queries, provided the query load is fairly homogeneous. They are edge-of-application-server if they materialize local database queries [Lu02] or implement mapping functionality, e.g., object-relational mapping as in J2EE container-based persistency. Integrated solutions are those where application data is cached through runtime objects designed by the application developer, as in J2EE bean-based persistency.

At the upper end, Web page caching usually is integrated into the WWW server, i.e., not directly accessible by the client through a separate service. Beyond the simple task of caching static pages, there are many approaches for caching dynamically generated Web pages. Often, edge-of-net caches known as Web proxy caches are added to the connection between WWW client and WWW server.

Novel challenges arise for the connection between WWW server and application server. Whereas database servers and Web servers offer a fairly narrow generic functionality, application servers may offer extensive and complex services that vary widely across servers. Consequently, all communication is via service function calls, e.g., in case of object-orientation via method calls. Since there is little likelihood that method calls are shared across different Web servers nor across different applications on the same Web server, the most natural solution seems one of cache integration into the Web server (method cache) [PW02].

7.3.2 The curse of consistency

Our notion of consistency refers to interpretability of the database as a possible state of the environment. This leaves consistency in the hands of the database server in our tiered architecture. If the database schema remains the same across all servers the meaning of the data is preserved across them. Conversely, if mappings take place in the caches the consistency may become endangered. In general, mappings are no longer generic so that an orthogonal solution along the lines of Section 4.1.3 seems more difficult to achieve.

A stricter notion associates consistency with the current state of the environment and enforces it by transactions. As data move upwards from peripheral storage across the

caches where they become replicated and rearranged according to the staging strategy, they become progressively more removed from the database state. Just consider that changes on the upper tiers propagate downwards with certain delays. The effect is known as the *cache coherency* or cache invalidation problem. If we followed the orthogonal approach of Section 4.1.4 we would have to add a global transaction manager to Figure 12 as a new, separate component. Indeed, such a solution is part of distributed systems or of middleware systems. On the downside, distributed transactions have a negative effect on both, performance and component autonomy.

If one is willing to deviate from strict consistency to a certain degree, solutions that control the propagation of changes downwards to the database server and from there upwards to other servers come into play. Examples are push versus pull strategies, synchronous versus asynchronous propagation, automatic refresh. Migrated caches seem to offer advantages over integrated caches because one may specify and enforce strategies without knowledge of and effects on the inner working of the servers.

A somewhat bitter conclusion is that in tiered architectures, as opposed to layered architectures, QoS parameters show considerable interdependence, i.e., less orthogonality. The optimum across all parameters is much more tedious to find and requires repeated iterations.

7.3.3 Adding robustness

In Section 4.1.4 robustness was defined in terms of transactions. As such it depends on strict consistency. Strict consistency, as we just noted, must be enforced by distributed and perhaps longer-duration transactions. As before, if one is willing to relax robustness and thus can do without a global transaction manager, local techniques for non-repudiation or irrevocability as in e-commerce take the place of strict consistency. On the other hand, since consistency relates solely to the database server, the caches may serve as a log for semantically richer transactions.

We observe again that robustness can not always be treated as an orthogonal quality because each connection between components may be individually affected.

7.3.4 Privacy protection

In Section 5.1 we viewed privacy protection through the concept of purpose and offered a solution for the database server that ensured that only those data were accessible whose purpose fitted the purpose of the query. Let's continue the speculation from there. We note that in the hypothetical architecture of Figure 7 data had already to be replicated on the upper layers. Hence, the caches of Figure 12 seem to be ideally suited to carry the quality of purpose up to higher tiers as purpose becomes redefined in a broader context. At the same time, though, Figure 7 introduced a firewall to separate the single query purpose from the data with their many purposes. Likewise, each connection would have to include a firewall to separate purposes.

8 Conclusion

Did we meet our self-imposed challenge to demonstrate that architectural design can be treated with some scientific rigor, that it can be made to follow a methodology that can be organized around well-defined criteria? We must leave it up to the reader to decide. By developing a design hypothesis and applying it mostly to established architectures a posteriori, we have so far only circumstantial evidence that the methodology is up to the task. Part of the evidence is that by changing the weight of the design criteria novel architectures may evolve. Even more striking is the fact that methodical design may reveal larger design spaces than initially conceived.

The design hypothesis itself revolves around the concept of service and service hierarchies. Services are described in terms of functionality and qualities. The hypothesis attempts to construct service hierarchies by decomposing service functionality under the guidance of a service quality chosen as the prime quality. The expectation is that service qualities are sufficiently orthogonal so that others can be taken care of simply by local additions or separate components. We could demonstrate that the method holds up reasonably well for layered architectures.

For component architectures where service qualities can only be controlled by proper choice of components and by regulating the connections between them we had to modify the hypothesis. We demonstrated why we suspect that with the smaller range of options we have to sacrifice the orthogonality of qualities, resulting in more iterations to arrive at a design that meets given criteria. Nonetheless, even in this situation the service philosophy seems to provide a suitable framework.

References

[Ab03] Abeck, S., Lockemann, P.C., Schiller, J., Seitz, J.: Verteilte Informationssysteme – Integration von Datenübertragungstechnik und Datenbanktechnik. dpunkt.verlag. 2003 (in German)

[Ag02] Agrawal, R.; Kiernan, J.; Srikant, R.; Xu, Y.: Hippocratic databases. Proc. 28th Intnl. VLDB Conference. 2002

[As79] Astrahan, M.M., et al.: System R: A relational database management system. IEEE Computer. 12:5. 1979. 42-48

[Ch81] Chamberlin, D.D., et al.: A history and evaluation of System R. Comm. ACM 24:10. 1981. 632-646

[He96] Herzog, U.: Effiziente Konsistenzprüfung in Datenbanksystemen. Infix. 1996 (in German)

[HR99] Härder, T.; Rahm, E.: Datenbanksysteme: Konzepte und Techniken der Implementierung. Springer, 1999 (in German)

[Lu02] Luo, Q.; Krishnamurthy, S.; Mohan, C.; Pirahesh, H.; Woo, H.; Lindsay, B.G.; Naughton, J.F.: Middle-Tier Database Caching for e-Business. Proc. ACM SIGMOD Conference. 2002. 600-611

[Mo01] Mohan, C.: Caching Technologies for Web Applications. Tutorial, VLDB 2001. http://www.almaden.ibm/u/mohan/Caching_VLDB2001.pdf

[PW02] Pfeifer, D.; Wu, Z.: A transparent client-side caching approach for application server systems. Submitted for publication. 2002

[St02] Starke, G.: Effektive Software-Architekturen – Ein praktischer Leitfaden. Carl Hanser. 2002 (in German)

[We88] Weikum, G.: Transaktionen in Datenbanksystemen. Addison-Wesley. 1988 (in German)

Wissenschaftliches Programm

Executing Nested Queries

Goetz Graefe
Microsoft Corporation
Redmond, WA 98052-6399

Abstract: *Optimization* of nested queries, in particular finding equivalent "flattened" queries for queries that employ the SQL sub-query construct, has been researched extensively. In contrast, with the exception of nested loops join, *execution* of nested plans has found little interest. Nested execution plans may result from a failure to flatten nested SQL expressions but just as likely are created by a query optimizer to exploit all available indexes as effectively as possible. In fact, if materialized views and index tuning perform as expected, few queries should require large operations such as parallel scans, sorts and hash joins, and most actual query plans will rely entirely on navigating indexes on tables and views. Note that only index navigation plans scale truly gracefully, i.e., perform equally well on large and on very large databases, whereas sorting and hashing scale at best linearly. Since a typical index navigation plan employs nested iteration, this paper describes techniques to execute such plans efficiently as well as means to cleanly implement these techniques. Taken together, these techniques can improve query performance by orders of magnitude, giving them obvious practical importance.

1 Introduction

Database and disk sizes continue to grow fast, whereas disk access performance and disk bandwidth improve much more slowly. If for no other reason than that, research into database query processing must refocus on algorithms that grow linearly not with the database size but with the query result size. These algorithms and query execution plans rely very heavily on index navigation, i.e., they start with a constant given in the query predicate, find some records in an index, extract further column values from those records, find more records in another index, and so on. The cost of this type of query plan grows linearly with the number of records involved, which might very well mean that the cost is effectively independent of the database size. Actually, the cost of index lookups in traditional B-tree indexes grows logarithmically with the database size, meaning the cost doubles as a table grows from 1,000 to 1,000,000 records, and doubles again from 1,000,000 to 1,000,000,000,000 records. The cost barely changes from 1,000,000 to 2,000,000 records, whereas the cost of sort or hash operations doubles. Moreover, it is well known that scanning "cleans" the I/O buffer of all useful pages, unless the replacement policy is programmed to recognize scans [S 81]. It is not likely, however, that CPU caches and their replacement policy recognize scans; thus, large scans will repeatedly clear out all CPU caches, even level-2 and level-3 caches of multiple megabytes. Based on these behaviors, on the growth rates of disk sizes and disk bandwidths, and on the recent addition of materialized and indexed views to mainstream relational database systems, we should expect a strong resurgence of index-based query execution and thus research interest in execution plans relying heavily on nested iteration.

Figure 1 shows a query evaluation plan for a simple join of three tables, with the "outer" input of nested iterations shown as the left input. The nested iterations are simple Cartesian products here, as the join predicates have been pushed down into the inner query plans. If the filter operations are actually index searches and if table T0 is small relative to T1 and T2, this plan can be much more efficient than any plan using merge join or hash join. An alternative plan uses multiple branches of nested iteration rather than multiple levels as the plan in Figure 1; this plan would be left-deep rather than right-deep. Of course, complex plans might combine multiple levels and multiple branches at any level.

Nested loops join is the simplest form of nested iteration. There is no limit to the complexity of the inner query. It might require searching a non-clustered index and subsequently fetching from the main table; intersecting two or more non-clustered indexes using bitmap, sort, or hash operations; joining multiple tables each with its own access paths; multiple levels of nested iteration with multiple sub-branches at each level; etc. For all but the most trivial cases, there are complex issues that need to be addressed in a commercial product and in a complete research prototype. These issues exist at the levels of both policies and mechanisms, with avoiding I/O or exploiting asynchronous I/O as well as with managing resources such as memory and threads.

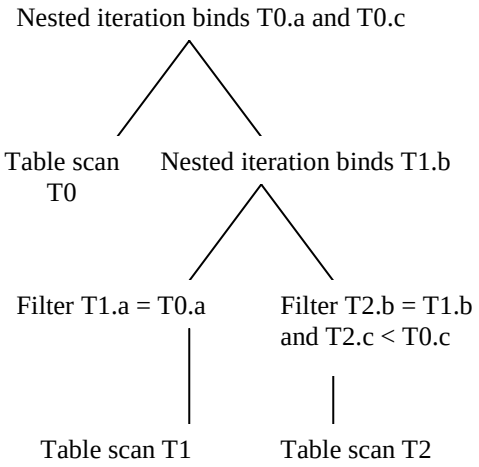


Figure 1. A query execution plan with moderately complex nested iteration.

The purpose of this paper is to explore these issues, to describe proven solutions and promising approaches, to detail how to implement those solutions, and to focus and stimulate research into open issues. Many useful techniques have long been known and are surveyed here, some for the first time. Other approaches and solutions are original contributions, e.g., the use of merged indexes for caching in nested iteration or how to manage batched correlation bindings in the outer and inner plans. For all of those techniques, data flow and control flow are detailed, including new iterator methods for clean and extensible implementations of the described techniques in sequential and parallel execution plans. The following section sets the stage for the remaining ones, which cover, in this order, asynchronous I/O, avoiding I/O through caching and other means, data flow, and control flow.

2 Background, scope, and assumptions

In the following discussions, we do not consider query optimization, e.g., "flattening" or "unnesting" or "decorrelating" nested query expressions into ones that do not employ nesting. While an interesting and traditionally challenging topic of research and commercial development, we consider it a complement to the present paper. Similarly, we presume here that the physical database design is given and cannot be tuned. Again, this is an important and challenging complementary topic in itself. We also do not describe algorithms or data structures for indexes, as those issues are covered elsewhere.

We presume typical contemporary database server environments, i.e., a large number of user connections and concurrent transactions; a small or moderate number of CPUs, each with its own cache; a fair amount of main memory divided among many connections, queries, operators, and threads; a number of disks larger than the number of CPUs by a small factor, say ten times more disk drives than CPUs; a database on disk many times larger than main memory; some virtualization in the disk layer such as caches in disk controllers, system-area networks, and network-attached storage; many tables of varying sizes, each with a small number of B-tree indexes on a few columns; non-clustered indexes pointing into heap files using record identifiers (page number and slot number) or into clustered indexes using unique search keys; etc. Without doubt, some of the techniques described here need to be adjusted if the environment changes drastically, e.g., if the entire database resides in main memory or if all indexes use hashing rather than B-trees.

For the sake of completeness, we very briefly review the various forms of nested loops join. *Naïve nested loops join* performs a simple scan of the inner input for each row of the outer input. If the outer and inner inputs are stored tables, *block nested loops join* actually employs four nested loops: the two outer-most loops iterate over pages (or blocks of some size) of the outer and inner inputs, and the two inner-most loops perform naïve nested loops among the rows in a pair of blocks. I/O volume is reduced by the blocking factor of the outer, i.e., the number of records in a block of the outer input, but the join predicate is evaluated just as often as in naïve nested loops. I/O will be slightly reduced if scans of the inner input go forward and backward, because of buffer effects each time the direction is reversed [K 80].

However, unless the inner input is very small, e.g., less than a few dozen rows, the only version of nested loops join used in practice is index nested loops join. Many query optimizers consider temporary indexes for the inner input, if no useful permanent index is available. These indexes are used only during a single execution of a specific query plan, and their contents are usually more restrictive than an entire stored table. The type of index does not really matter – it might be a B-tree index, a hash index, a bitmap index, a multi-dimensional index, etc. – as long as the index supports the join predicate efficiently and effectively, i.e., without fetching many more index entries than truly necessary.

In general, nested iteration works better than set-oriented algorithms such as hash join if the outer input is very small relative to the database, such that navigating the inner input using indexes leaves most of the database untouched. Typical queries include not only OLTP requests but also cursor operations such as "fetch N more result rows" and their internet-age equivalents for "next page" queries with their typical "top" operations. In all

these cursor-like operations, resuming where the prior request left off requires sort order, and nested iteration preserves the sort order from the outer input, and it might even extend it with additional ordering columns from the inner input.

A special case of a small outer input is a set of parameters left unbound in the query text, to be bound when an application invokes the query. This situation can be modeled, both in optimization and in execution, as a nested query. The outer input is a single row with a column for each parameter, and the inner input is the entire original query. This is one of the few cases in which a query optimizer can truly rely on its "estimate" for the size of the outer input (other cases include exact-match predicates on unique columns and range selections on unique integer columns). If a parameterized query is optimized and executed as a nested query, it is easy to see how "array invocations" with multiple sets of parameters can be supported.

Even for large queries, nested iteration can be more efficient than merge joins or hash joins, in particular if covering indexes are available, i.e., indexes that deliver all columns required in a given query without fetching rows from the base table. Consider a self-join of a table *order details* on the (non-unique) column *order key*, e.g., for an analysis of purchasing combinations. A hash join needs to scan the entire table twice, with no beneficial buffer effects if the table size exceeds the buffer size, and it requires a massive amount of memory as well as additional I/O for overflow files. In a merge join of two pre-sorted inputs, the two scans benefit from sharing pages in the I/O buffer, but the merge join requires support for the backup logic of many-to-many merge joins, often implemented by copying all records from the inner input into the local buffer. An index nested loops join, however, benefits from the I/O buffer such that all data pages need to be read only once, and it exploits the index on *order key* to find matching inner rows for each outer row. In general, in a well indexed database, the disk access patterns of index nested loops join and merge join are often very similar [GLS 93], and index nested loops join performs as well as or better than merge join.

DeWitt et al. [DNB 93] found that index nested loops join can best other joins, even on parallel database servers considered by many showcases for hash-based join algorithms. The essence is that I/O cost often dominates join cost, and that index nested loops performs well if the index on the inner input can ensure that only a fraction of the inner input must be read from disk. In fact, it has to be a small fraction, since sequential I/O (as used for hash join) is much more efficient than random I/O (as used by index nested loops join), in particular if asynchronous read-ahead overlaps processing time and CPU time.

3 Asynchronous I/O

Asynchronous I/O is, however, not limited to sequential scans. In a naïve implementation of index nested loops, each index lookup and record fetch is completed before the next one is initiated. Thus, there must be at least as many threads as disks in the system, because otherwise some disks would always be idle. This would not be good given that disk I/O is a precious resource in database query processing.

Index nested loops join can, however, if implemented correctly, exploit asynchronous I/O very effectively. In some recent IBM products, unresolved record identifiers are gathered into fixed-sized lists, and I/O is started concurrently for all elements in the list once the

list is filled up. Some recent Microsoft products keep the number of unresolved record identifiers constant – whenever one record has been fetched, another unresolved record identifier is hinted to the I/O subsystem. In general, steady activity is more efficient than bursts of activity.

Fetch operations can exploit asynchronous I/O for heap files, when unresolved record identifiers include page numbers, as well as for B-tree indexes, where unresolved references are actually search keys. There are two effective methods for combining synchronous and asynchronous I/O in those situations, both relying on a *hint* preceding an *absolute* request for the desired page. First, while descending the B-tree searching a hinted key, the first buffer fault can issue an asynchronous read request and stop processing the hint, which will be resolved later using synchronous I/O for each remaining buffer fault. Second, presuming that in an active index practically all pages above the leaf pages will reside in the buffer, searching for a hinted key always uses synchronous I/O for the very few buffer faults for nodes above leaves, and always issues an asynchronous I/O request for the leaf page. In practice, the two methods probably perform very similarly.

The generalization of such asynchronous I/O for complex objects was described and evaluated by Keller et al. and Blakeley et al. [KGM 91, BMG 93], but not adopted in any production system, to the best of our knowledge. The central idea was to buffer a set of object roots or root components such that the I/O system always had a substantial set of unresolved references. These would be resolved using a priority queue, thus attempting to optimize seek operations on disk. If object references are logical, i.e., they require mapping to a physical address using a lookup table, both lookups should use asynchronous I/O.

Always keeping multiple concurrent I/Os active can be a huge win. Most server machines have many more disk arms than CPUs, often by an order of magnitude. Thus, keeping all disk arms active at all times can improve the performance of index nested loops join by an order of magnitude. Even in a multi-user system, where many concurrent users might keep all resources utilized, reduced response times lead to higher system throughput due to reduced lock contention. Keeping many disks busy for a single query requires either aggressive asynchronous I/O or many threads waiting for synchronous I/O. For example, in some Informix (now IBM) products, the number of parallel threads used to be determined by the number of disks, for precisely this reason. It is probably more efficient to limit the number of threads by the number of CPUs and to employ asynchronous I/O than to use very many threads and only synchronous I/O, since many threads can lead to excessive context switching in the CPUs, and more importantly in the CPUs' data caches. This presumes, of course, that I/O completion is signaled and not actively and wastefully polled for.

4 Avoiding I/O

As attractive as it is to speed up I/O, it is even better to avoid it altogether. The obvious means to do so is caching, e.g., in the file system's or the database manager's buffer. When computing complex nested queries, however, it is often beneficial to cache results even if they may not remain in the I/O buffer and may require I/O, because I/O for the cache might be substantially less than I/O to recompute the cache contents. Note that caching avoids not only repetitive computation but also repetitive invocations of the lock

manager, and that the I/O buffer usually can retain cached results more densely than the underlying persistent data. Caching has been used in several products, e.g., IBM's DB2 for MVS [GLS 93] as well as recent Microsoft database products.

It is well known that the result of an inner query can be cached and reused if the inner query does not have any references to outer correlation values, i.e., the inner query is equivalent to a constant function. Since this is the easy case with a fairly obvious implementation, we do not discuss it further here. Similarly, in a nested iteration where the outer input is guaranteed to have only a single row, caching is not very difficult or interesting (at least not caching within a single query).

Expensive user-defined functions are closely related to nested queries; in fact, since either can invoke the other, it is a chicken-and-egg problem to decide which one is a generalization of the other. In research and in practical discussions, there is usually a difference in emphasis, however. First, functions are usually presumed to invoke user-defined code with unpredictable execution costs, whereas nested queries usually perform database searches, where standard selectivity estimation and cost calculation apply, with buffer (caching) effects as added complexity. Second, functions usually are presumed to have scalar results, i.e., a single value, whereas nested queries often produce sets of rows.

4.1 Caching results of the inner query

For a cache, any associative structure such as a hash table or a B-tree index at the root of an inner query plan is sufficient. The search key is the correlation value, and the retrieved information is the result of the inner query for that correlation value. A single simple lookup structure permits caching scalar values, e.g., a sum, an average, or a Boolean flag whether or not the result of the inner query is empty or unique, i.e., free of duplicate rows [HN 96]. If the result is a set of rows, it might be necessary to employ two lookup structures, one index to indicate whether a correlation value has occurred before and another one with the actual results. If only one index is used, it is not clear how to remember that some correlation value produced an empty set. The first index, which we may call the control index, could be a bitmap index, but it could not be a standard bitmap filter, since bitmap filters usually permit hash collisions. It could be restricted to contain only those outer correlation values that resulted in an empty inner result, or it could contain all outer correlation values already seen.

If the outer input binds multiple correlation columns as parameters of the inner query, the search key of the cache structure contains multiple search columns. If different branches of the inner query's execution plan use different subsets of these correlation values, it is also possible to cache not at the root of the inner query but at the root of those branches. The advantage is that each of the branches is executed fewer times; the disadvantage is that the computation between the roots of the branches and the entire inner query is executed multiple times. In extreme cases, it is possible to cache in multiple places, but considering too many combinations of caches can result in excessively complex and expensive query optimization.

For example, consider the query execution plan in Figure 2, which differs from Figure 1 as there is only one nested iteration and a join or intersection within the inner plan. Note that tables T1 and T2 could also be different indexes of the same table in a plan using

index intersection for a conjunctive search predicate. Point A is the root of the inner plan and a good place to cache inner results. However, it might also be possible to cache at points B or C, or even at all three points. If T0 contains 10,000 rows, T1 and T2 are scanned 10,000 times each, unless caching is employed. If T0 contains only 1,000 distinct combinations of the pair (a, b), caching only at point A means that T1 and T2 are scanned 1,000 times each. If, on the other hand, T0 contains only 100 distinct values of (a) and only 200 values of (b), caching at points B and C means that T1 and T2 are scanned only 100 and 200 times, respectively, although the intersection operation is still performed 10,000 times. Caching at all three points ensures both the lowest counts of intersection operations (1,000) and of scans for T1 and T2 (100 and 200).

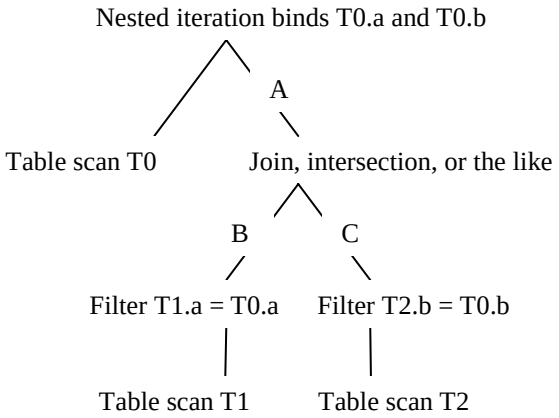


Figure 2. A query execution plan with moderately complex nested iteration.

Many query optimizers will consider the costs and benefits of caching only at point A for a query of this type. Otherwise, the optimization time becomes excessive – even for this simple example, there are 2^3 alternative caching strategies to consider. Moreover, different rows in T0 with different values in (a) and (b) may result in different selectivities in the filters on T1 and T2, and the chosen caching strategy must work well for all of those. Fortunately, in most cases, the inner query plan relies on index searches rather than table scans, and indexes can create powerful cache effects of their own.

For example, if T1 is searched with an index on (a), the index itself serves the role of a cache at point B. Moreover, one can sort the outer input on the correlation values, such that equal values are in immediate succession, and inner query results can be reused instantly. By sorting the outer input T0 on (a) or by scanning an index on T0.a, the index searches for T1 will be very "orderly," creating a disk access pattern quite similar to a merge join of T0 and T1. If multiple rows from a single leaf page within the index on T1.a participate in the query result, it is guaranteed that this leaf will be fetched only once, independent of the size of the buffer pool and its replacement policy. However, if there are multiple correlation columns as in this example, it is not clear on which column sequence to sort. In the example above, should the optimizer elect to sort the outer input T0 on (a), on (a, b), on (b), or on (b, a)? If input T0 has an index only on (a) but not on (a, b), is an explicit sort operation worthwhile? Quite obviously, only a cost-based query optimizer can answer such questions, case by case.

There is a related question about sorting the result of searching an index within the inner query. If there is a useful index, say on column (a) of table T1, how is the result cached? If the join operation between T1 and T2 uses a predicate on column (c) in both T1 and T2, the join can be computed using a fast merge join without sorting if, for any value of the pair (a, b) from T0, the caches at points B and C retain values sorted on (a, c) and (b, c), respectively. In other words, the sort on (c) is part of the computation cached at points B and C, enforced prior to caching intermediate results and not repeated when retrieving data from the caches. Using the specific counters assumed earlier, the merge join might be executed 1,000 or 10,000 times depending on caching at point A, but there will be only 100 sort operations for rows from T1 and 200 sort operations for rows from T2. In this case, the benefit of the caches at B and C could be savings in sort effort, whether or not there are worthwhile savings in I/O for the permanent data.

Unfortunately, if a cache grows large, it may cause I/O cost of its own. In that case, it is possible to limit the total size of a cache by subjecting its entries to a replacement policy. Units of insertion and deletion from the cache are correlation values from the outer input of the nested iteration. Promising replacement policies are LRU (least recently used) and LFU (least frequently used); a reasonable measure of "recent" can be a sequence number attached to the outer input (using a "rank" operation, if available). Both policies can be augmented with the size of an entry, i.e., the size of an inner result for an outer row, and the cost of obtaining it, if this cost differs among outer rows. If the cache is represented in two indexes, one containing only previously encountered outer correlation values and one containing result sets from the inner query, information on costs and usage belong into the first one. The replacement policies for the two indexes do not necessarily have to be the same, such that statistics might be kept even for outer correlation values that have temporarily been evicted from the cache of inner results – reminiscent of the LRU-k policy that has been proposed for disk buffers [OOW 93].

Materialized and indexed views can be exploited as special forms of the caches discussed so far. If incremental maintenance of materialized views with join and semi-join operations is supported, permanent auxiliary tables can perform the role of control indexes discussed above. For any value currently found in the control table, the materialized view contains the result of the nested query. Creation of materialized views and their control tables can be left to an index tuning tool or perhaps even the query optimizer. Query execution can simply ensure that the actual parameter values are present in the control table and then immediately retrieve the results rows from the result table. "Ensuring" presence of certain rows in the control table uses the "merge" operation of ANSI SQL, sometimes also called "insert or update," and its implementation in query execution plans. If control rows contain information beyond the actual parameter values, e.g., last-used time stamp for use by an LRU replacement scheme, there is indeed an update component. The transactional context for inserting a new row into the control table should be a system transaction, not the user transaction and its isolation level etc., although this system transaction should run within the user's thread (similar to a system transaction that splits a B-tree page). The user transaction then locks the control rows it reads, precisely the same way as rows in an ordinary table or in a materialized view. An asynchronous system task can remove rows from the control table that are not worth keeping, which of course instantly removes the appropriate result rows from the materialized view and all its indexes.

4.2 Implementation of sorting

Sorting the outer input is worthwhile with respect to I/O in the inner plan if it can achieve advantageous buffer effects for page accesses in the inner plan. If a table or index accessed in the inner plan is so small that it will fit entirely in memory or in the I/O buffer, sorting the outer does not reduce I/O in the inner. Conversely, if the table or index accessed in the inner is so large that there are more pages (or whatever the unit of I/O is) than there are rows in the outer input, sorting the outer does not reduce I/O in the inner, except if the outer input contains rows with duplicate correlation values.

In order to achieve the benefits of sorting with the least run-time cost, the implementation of sorting must be integrated neatly with the query processor. Intermediate result rows on the input of the sort operation must be pipelined from the prior query operation directly into the sort, and sorted rows must be pipelined directly into the next operation. In other words, the only I/O due to a sort operation should be sorted intermediate run files. A sort implementation that is limited to reading its input from a stored table or index and writing its output to a stored table or index may triple the I/O cost due to the sort. Nonetheless, despite just such an implementation of sorting, DeWitt et al. observed substantial performance advantages for nested loops join due to sorting the outer input [DNB 93].

Rather than sorting the outer input completely, sorting only opportunistically might achieve sufficient locality in the inner input to realize most of the potential savings. For example, if the cost of an external merge sort is high, the optimizer might choose to run the outer input through the run generation part of an external merge sort but without writing any runs to disk. This technique is promising if the number of records in the generated runs is larger than the number of pages in the file from which matching records are fetched. In addition, if the inner input is stored in a clustered index, sorting the outer might improve cache locality while searching the clustered index. In order to preserve a steady flow of intermediate result records within the query plan, replacement selection based on a priority queue is probably a better algorithm than read-sort-write cycles typically associated with quicksort. Moreover, replacement selection on average yields runs twice as long as quicksort.

4.3 Merged indexes

Another means to avoid I/O is physical clustering. Master-detail clustering in relational and other database systems has long been known to reduce I/O when nested iteration plans assemble complex objects [H 78].

As a prototypical example, consider how to merge the clustered indexes for the relational tables *customers*, *orders*, *order details*, *invoices*, and *invoice details*. The goal is that all information about a single customer is in a single leaf page or in adjacent leaf pages. The significant characteristics of the example are that it includes both multiple levels and multiple branches at one level, and that it mixes “strong entities” and “weak entities”, i.e., ones with their own unique identifier, e.g., *order number*, and ones which rely for their existence and identity on other entities, e.g., *line number* within an *order*.

One very useful way to think about master-detail clustering is as multiple traditional single-table indexes merged into a single B-tree (or hash structure, or B-tree on hash values). In this design, all participating single-table indexes share some leading key columns, but

typically not all key columns. Alternatively, one can extend some or all of the participating record types such that each record contains the union of all key columns. However, this alternative suffers from poor extensibility, because all existing records must be reformatted when a new single-table index and a new record type with a new key column are introduced, even if the new table contains no rows yet.

In the more flexible design, the sort key in a merged B-tree is more complex than in a traditional single-index B-tree. Each column value is prefixed by its domain (or domain identifier, or type, or type identifier), very similar to the well known notions of tagged union types. A new column type is the identifier of the single-table index to which a record belongs. Columns following the index identifier may omit their type identifiers. Figure 3 shows an example record for the *orders* table. While the design in Figure 3 lends itself to key normalization of the entire record, alternative designs tag each record with the identifier of its schema and compare records by alternating between two schemas and the two actual records.

Even the index of the *order details* table must include the leading field *customer number* in order to achieve the desired clustering of records in the B-tree. Such denormalization and field replication is required if two conditions hold. First, there must be multiple levels in the join graph that reassembles the complex objects from the index. Second, some intermediate tables must have unique keys of their own (a primary key that does not include a foreign key, or strong entities in the entity-relationship model) such as the table *orders*. In effect, the index for *order details* could be thought of as an index not on the table *order details* but as an index on the view joining *orders* and *order details*, and the maintenance of the index uses standard update plans for indexed (materialized) views.

Field value	**Field type**
"Customer number"	Domain identifier
Customer number	Actual value
"Order number"	Domain identifier
Order number	Actual value
"Table and index identifier"	A fixed domain identifier
Orders table, Customer Order index	References to catalog entries
Order date	Actual value
Order priority	Actual value
…	…

Figure 3. A sample record in a merged B-tree index.

Note also that the above index does not permit searching for *order* or *order details* rows by their *order number* only. Thus, it might be useful to introduce a separate index that maps *order numbers* to *customer numbers*. Moreover, either such an index is required for both the *orders* and the *order details* tables, or the query optimizer can exploit the index on *orders* even for *order details* searches on *order number* by analyzing primary and foreign key constraints, i.e., functional dependencies within and across tables.

Implementing master-detail clustering as multiple traditional single-table indexes merged into a single B-tree creates useful flexibility. For example, in a many-to-many relationship such as between suppliers and parts, it is possible to define two indexes on the table

representing the relationship, *part supply*, one on the foreign key *part number* and one on the foreign key *supplier number*, and then merge one of those indexes with the primary key index on the *parts* table and one with the primary key index on the *supplier* table. This design enables fast navigation from parts to suppliers and from suppliers to parts.

Even multiple indexes from a single table can be merged into a single B-tree, e.g., for a recursive relationship such as "reports to" among employees. In this example, the two indexes would be on *employee id* and *manager's employee id* – one of these could even be the clustered index, if desired.

In addition to I/O savings when assembling complex objects, merged indexes are also very useful for caching in query plans with nested iteration. In the earlier discussion of caches and replacement policies, we presumed two indexes, a control index that captures which combinations of parameter values have been processed before, and a data index that contains the results of nested query executions. Since both indexes use the correlation columns as the search key, it is obvious that they can be merged into a single B-tree, with equally obvious I/O savings.

4.4 Index intersection versus multi-dimensional indexes

For equality predicates on multiple columns, single-dimensional multi-column indexes are sufficient. For example, the predicate *where salary = 20 and bonus = 5* can be processed efficiently using a traditional multi-column B-tree on *employee (salary, bonus)*. If range predicates on multiple columns are frequent, e.g., *where salary between 20 and 30 and bonus between 3 and 9*, multi-dimensional index could be considered. Rather than implementing an entirely new index structure, a single-dimensional index combined with a space-filling curve can be very attractive, e.g., a B-tree for a Peano or Hilbert curve [B 97, RM 00, MJ 01].

There are effective techniques to reduce the I/O even in a single-dimensional multi-column index. For example, if there are only a few qualifying values of the first column (*salary* in the example index above), these values can be enumerated efficiently using the index, and then for each such value, index entries that qualify for the entire predicate can be searched for directly. This technique is exploited in some products by Tandem (now HP) and Sybase [LJB 95].

5 Data flow

The preceding discussion covered I/O, both avoiding it and speeding it up when it cannot be avoided. The present section covers data flow, i.e., how records should move in a complex query execution plan to achieve the desired effects on I/O. The subsequent section will cover control flow, i.e., how the desired data flow can be implemented with efficient and extensible mechanisms in single-threaded and multi-threaded query execution.

5.1 Batches of bindings

Sorting the outer input is only one possible method of many for improving the locality of data access in the inner plan. An alternative is to loosen the restriction that the inner execution plan can execute on behalf of only one outer row at a time. In other words, multi-

ple outer rows are bound and the inner plan executes only once for the entire set, hopefully improving execution efficiency. In general, it seems that batched execution is readily applicable if all intermediate results in the inner plan either are the same for all outer bindings in the batch or if distinct bindings in the outer bindings partition all stored tables and intermediate results in the inner plan into disjoint subsets of rows. In the first of those cases, the expected efficiencies are similar to caching; in the second case, they are similar to the advantages of set-oriented SQL queries compared to application-level row-at-a-time cursor operations. The size of the set may be determined simply by counting outer rows or by some column value in the outer input, very similar to value packets [K 80a].

5.1.1 Temporary storage for batches

In order to achieve these efficiencies, the batch of outer bindings must be accumulated and thus stored within the inner plan. Thus, it might be useful to introduce a new leaf in the inner plan (or multiple copies of the same leaf). This leaf is similar to a table stored in the database, except that it is a temporary table populated from above with a batch of outer bindings. If such a leaf is the direct input into a sort or (as build input) into a hash join, special implementations or modes of those operations may subsume the function of the new leaf. Thus, a sort operation may be the leaf of a query plan, with its input passed in as a batch of outer bindings; and a hash join may have only one input (the probe input).

For example, consider an inner plan that includes a filter operator with a predicate like "t.a = @a" where "t" is a table scanned in the inner plan and "@a" is a correlation bound by a nested iteration above the filter operator. For this filter to work for a batch of values for "@a", it might be advantageous to put all those values into a hash table. Thus, the filter becomes a hash join with the hash table built from outer correlation bindings. Note that this is very similar to a hash-based implementation of the filter operation for predicates of the form “t.a in (@a, @b, @c, @d, …, @z)” – some applications, in particular those with their own caches, employ very long “in” lists that benefit from a hash table in the filter operation, and a hash-based implementation of “in” filters is required to support those applications efficiently. Rather than implementing multiple forms of filter, it might be simpler to employ the hash join implementation for all filters that require memory for binding values; in the extreme case, with a single row in the build input.

Alternatively, implementation effort is reduced by implementing accumulation of outer bindings only in the new leaf operator and, where necessary, to use the leaf operator as the input for sort and hash join, e.g., as build input in the example. Using this simplification, both sort and hash join implementations are less complex and more maintainable, at probably only a moderate loss of execution efficiency.

One of the possible inefficiencies in an execution plan with batched bindings is that the outer rows must be visited twice within the outer plan, once to bind correlation values for the inner query and once to match the result of the inner query with the outer rows. Moreover, these rows might have to be stored twice, once in the outer execution plan and (at least) once in the inner execution plan. Fortunately, it is possible to share the actual store location or, alternatively, to bind entire outer rows such that the result of the inner query includes all columns required in the result of the nested iteration. The convenience and efficiency of storing and revisiting those outer rows might determine the optimal size of each batch of outer rows. One means to ease the implementation is to create a temporary

rary table, and to scan it repeatedly as required. If the batches are small, the temporary table will remain in the buffer pool and therefore never require I/O. If larger batches reduce more effectively the total cost of executing the inner plan and if those batches are large enough to spill from the buffer pool, one might as well compute and spool the entire outer input before initiating the inner input. This query execution plan achieves precisely the effect called *sideways information passing* or *magic decorrelation* in query optimization [SHP 96], which is closely related to semi-join reduction [BC 81].

5.1.2 Details of matching outer rows and inner results

If the inner query is invoked for batches of outer rows, each inner result row must be matched correctly with outer rows. If the outer correlation values may contain duplicates, either these duplicates must be removed prior to executing the inner query or they must be made unique by adding an artificial "rank" column. This notion of an artificial rank column is a standard technique when duplicate rows can create problems, e.g., when non-clustered indexes point to base rows in non-unique clustered indexes using search keys, when un-nesting and re-nesting non-first-normal-form relations [ÖW 92], when pausing and resuming a cursor or a series of next-page queries [L 01], etc.

The operation that matches outer rows and results from the inner query plan also needs to ensure correct scalar results. In SQL, a nested query may be used in place of a scalar value, e.g., "where T0.a = (select …)", as opposed to in place of a table expressions, e.g., "where T0.a in (select …)". In the latter case, any number of result rows from the inner query are permitted. In the former case, however, it is a run-time error if the inner query produces zero or multiple rows. These semantics must be preserved in any execution plan, including plans that employ batches of outer rows. Suitable algorithms are well known as "hybrid cache" [HN 96] or "flow distinct" [GBC 98].

5.1.3 Mixed batched and non-batched execution

If, in a complex inner plan, some operations permit batches and some do not, it is possible to invoke the entire inner plan with batches yet invoke some parts of the inner plan one outer row at a time. A simple technique to achieve this is adding to the inner plan a nested iteration (to bind one outer row at a time), with a leaf operator (to store an entire batch) as outer input and the row-at-a-time part of the inner plan as inner input.

Figure 4 illustrates this idea. It only shows the crucial components; a query optimizer would not choose this plan as it is as the inner plan is trivial. The upper nested iteration creates batches. Bindings are spooled in a temporary table in the leaf node. Once an entire batch has been bound, the upper nested iteration requests result rows from the lower one. To produce those, the lower iteration will obtain a single row from the spool operator, bind it to the inner plan, and pass result rows to the upper nested iteration.

5.2 Parallel execution

While batches can be very helpful for reducing I/O, they are practically required for efficient parallel execution of nested query plans, in particular nested query plans in distributed-memory parallel database servers. On the one hand, the cost of crossing boundaries between threads, processes, or machines is very high compared to invoking iterator

methods (a simple invocation of a virtual method). Thus, each crossing of boundaries should be leveraged for many rows. On the other hand, each thread executing the inner plan may be invoked by multiple threads executing the outer plan and controlling the nested iteration. Thus, inner result rows must be tagged with the outer row and the requesting outer thread. The required tags are very similar to the tags required in batched execution of inner plans; thus, any tagging required for executing inner plans for batches of outer rows introduces no additional overhead.

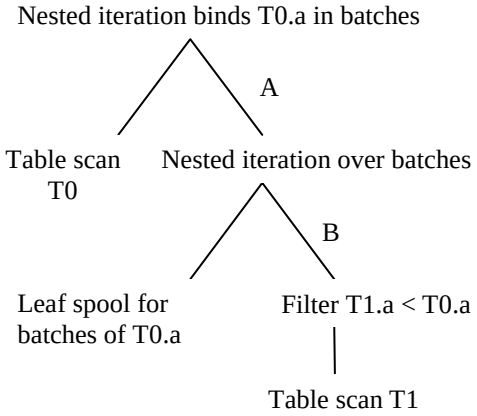


Figure 4. Using two nested iterations to create and dissect batches.

For these two reasons, inner query plans in parallel environments are best invoked for batches of outer rows. If the switch to batched execution cannot be accomplished automatically and transparently by the query execution engine, the query optimizer should attempt to rewrite the query using techniques described elsewhere as semi-join reduction and as magic decorrelation [BC 81, SHP 96]. If those techniques do not apply, batches of outer rows can be passed to the inner plan across thread, process, or machine boundaries, yet such batches are processed one row at a time using the technique shown in Figure 4 for complex inner plans in which some operations permit batch at a time operations and some do not.

6 Control flow

Given the large variety of algorithmic techniques described so far, implementing a state-of-the-art query execution engine might seem dauntingly complex. This section shows how to integrate the above techniques into an iterator interface very similar to the ones used in most commercial database systems [G 93]. The main operations for an iterator are, very similar to a file scan, *open*, *get next record*, and *close*. Each iterator invokes and therefore schedules all its input iterators or producers in demand-driven query execution. Alternatively, each iterator could invoke and schedule its output iterators or consumers in data-driven query execution, using an *accept next record* iterator method. Demand-driven query execution is preferable if multiple operators execute within a single process and if joins are more frequent than common subexpressions, i.e., when there are more operations with multiple inputs than operations with multiple outputs. On the other hand, data-

driven iterators seem preferable if iterators always communicate across thread boundaries or even machine boundaries and if common subexpressions are more frequent than joins.

6.1 Generalized spool iterator

In the following, we presume iterators based on demand-driven data flow. For common subexpressions, we presume a *spool* iterator that buffers its single input stream and provides the data to its multiple consumers at their desired pace. This operator exists in some form in most systems, for multiple purposes such as staging the result of a common subexpression or providing phase separation for protection from the Halloween problem [M 97]. As much as possible, in-memory buffers ought to be used, although temporary disk space must be available in case the input stream is very large and multiple consumers demand data at different rates. If the input data are sorted in a way that is useful later in the query plan, a sorted data structure such as a B-tree ought to be used.

The spool iterator can consume its input eagerly or lazily. The eager mode consumes its entire input immediately when *open*ed, similarly to a sort iterator. The lazy mode consumes its input only as fast as the fastest consumer requires spool output. The two modes can be a single implementation with a simple run-time switch.

This spool iterator can be generalized beyond these two prototypical modes for efficient nested iteration. For example, asynchronous I/O can be implemented using two fetch iterators, one providing prefetch hints to the store and the other actually obtaining the fetched records. These fetch iterators are separated by a spool iterator with a fixed-size record buffer to ensure that the store can always have multiple outstanding asynchronous I/O operations. This buffer can be run strictly first-in-first-out or it can use a priority queue (replacement selection) to pre-sort records. This mode of spool also can be useful in the outer input of a general nested iteration, as it may increase locality within the inner query without much cost. The buffer can be replenished from the input only when entirely empty (batch mode) or after each record passed to the consumer iterator (sliding window mode). A typical use case for the latter variant is enabling asynchronous I/O, while the former variant allows temporarily releasing resources in one part of a query plan for use in another part, e.g., releasing memory in an outer plan to make it available in an inner plan. However, managing resources explicitly and efficiently requires additional iterator methods to control resource usage, which will be discussed next.

6.2 New iterator methods

The principal mechanism to adapt the iterator interface to nested iteration is to extend the set of methods provided by all iterators. With almost half a dozen additional methods, a state diagram or a state transition table are the most concise means to convey all permissible method invocation sequences. A very helpful utility while developing a query execution engine is a “debug” iterator that can be inserted at any place in any query plan, where it enforces the state transition diagram, counts or prints intermediate result records, verifies sort order and other data properties predicted by the query optimizer, etc. If a version of this iterator is included in the final release of the query execution engine, it can compare anticipated and actual record counts or column distributions, and can thus also serve as the foundation for run-time feedback to the optimizer.

The required states capture whether an iterator is *open* or *closed*, whether or not all parameters are bound (*ready*), whether a result record has been produced but not yet released (*busy*), and whether or not the iterator is *paused*. Introducing the new iterator methods one by one will also clarify the purpose and semantics of these states.

Old state	Iterator method	New state	Comment
Closed	Open()	Open	
Open	Rewind()	Ready	Nothing unbound
Ready	GetNext()	Busy	Success
Ready	GetNext()	Ready	Failure, e.g., end
Busy	Release()	Ready	
Ready	Close()	Closed	
Ready	Rewind()	Ready	
Open	Close()	Closed	
Open	Bind()	Open	
Open	Unbind()	Open	
Ready	Unbind()	Open	
Ready	Pause()	Ready & paused	
Ready & paused	Resume()	Ready	
Busy	Pause()	Busy & paused	
Busy & paused	Resume()	Busy	
Ready & paused	Rewind()	Ready	
Ready & paused	Unbind()	Open	
Ready & paused	Close()	Closed	

Table 1. Iterator methods and state transitions.

In nested iteration, an additional iterator method *rewind* is required. For each outer row, the inner input is rewound, and the entire inner result can be recomputed. Most iterators implement this method by rewinding all their inputs; in this case, a rewind operation restarts an entire query plan. One exception to this rule is the *spool* iterator, which can simply fall back on its buffer to produce its output over and over. For simplicity, it might be tempting to require that the spool be eager; however, for efficiency, this is not a good design, because many nested queries terminate early, e.g., an “exists” or “top” subquery. Other “stop and go” operators might also implement efficient local rewind, e.g., sort and hash join. Unfortunately, hybrid hash join is more efficient than the original Grace hash join precisely because it does not write all its input to temporary files; thus, rewinding hybrid hashing might require more special cases than one is willing to implement, test, and support. In-memory hashing and completely external hashing (Grace hash join without dynamic destaging) lead more easily to efficient rewind operations. Sorting, on the other hand, can implement rewind operations quite readily; for example, only the final merge step needs to be restarted in an external merge sort.

For nested iteration with correlations, *bind* and *unbind* methods are required. If batched execution is not implemented, *bind* and *unbind* calls must be alternating strictly. In most cases, bind and unbind methods recursively traverse the entire inner plan, even across nested iteration operations within an inner plan. Implementations of the unbind method should not abandon all cached intermediate results, and the bind method should indicate

whether or not the new bindings invalidate intermediate results cached further up in the execution plan. In execution plans with complex inner plans, it is advantageous to annotate each sub-plan (or its root iterator) with the set of unbound parameters, and to avoid tree traversals where bind operations have no effect.

In order to implement batched execution, multiple *bind* operations without interleaved *unbind* operations must be permitted. In effect, a table of parameters is streaming down with the inner query plan, from the nested iteration towards the scan operations at the plan's leaves. The rows of this table, or the records in this data stream, may be buffered at one or multiple places in the inner plan, for later use. For example, an *exchange* operator might employ buffering, to be discussed shortly. As another example, a special spool iterator might be a plan leaf and buffer not an input stream but a table of parameters. During execution of the inner plan, the rows buffered in a spool iterator in *leaf* mode are like another input table for the inner plan – in fact, "magic" query execution and side-ways information passing presume that the entire set of parameters is a single large batch and serves as an additional input table in the inner query plan, which can readily be implemented using this *leaf* mode of spool.

In addition to the *close* method, which releases all resources but also terminates all iteration, a method is needed that lets a plan release resources for a while, in particular memory, yet later resume iteration. The methods to do so are *pause* and *resume*. For example, to pause a scan might release buffer pages used for double buffering, or a sort operation might write its input to disk even though the input was smaller than the memory available to the query. More interesting and more efficient, however, are iterators that do not immediate release their memory, but only register it as "waiting" with a query-wide memory manager. Only if the memory is indeed needed, and only as much as needed, will indeed be released. For example, if two sort operations feed a merge join, and their combined input is only slightly larger than the memory allocated to the query, only part of the first sort's input is written to a run file on disk, the remainder kept in memory as an in-memory run, and the two runs merged when sorted input is required by the merge join.

There are three typical use cases for the pause and resume methods. First and most importantly, a binary operation needs to open its inputs one at a time, and whichever input is opened first should pause while the second input is being opened. For example, a merge join fed by two sort operations can pause the first sort (which can therefore release its memory) while opening the second input. Earlier work presumed that between an iterator's *open* invocation and its first *get next* invocation would be the equivalent of a *pause*. Explicit iterator methods enable more control without requiring much more mechanism. The merge join in this example is typical for any bushy plan with resource contention among its branches.

Second, operations such as minor sort (sorting a pre-sorted stream on additional columns) or spool in batch mode consume their input and produce their output in batches. After a batch has been consumed, the input plan should release its resources if they can be used more productively elsewhere, e.g., in the inner plan while pausing the outer input. Thus, the spool iterator in batch mode should invoke the pause method on its input after filling its buffer with input rows.

Third, after a nested iteration iterator has bound a batch of outer rows to the inner query, the outer input may pause and release its resources for use other operations, in particular the inner plan. If both outer and inner plans are complex plans with memory-intensive bitmap, sort or hash operations, explicitly releasing and re-acquiring resources permits better performance with less memory contention.

Finally, as a performance feature, it may be useful to implement an iterator method that combines the bind-rewind-next-unbind sequence for a single record in a single method invocation, because there are many cases in actual query plans where it is known at compile-time that only a single record is needed. Typical examples include "exists" predicates and scalar nested queries, i.e., nested queries in place of scalar values. For the latter example, it is also very useful to implement a control flag in the iterator actually joining outer and inner rows that raises a run-time error if the inner result is empty or if it contains more than a single row (although the multiple-row case can be detected in a separate iterator, too, as discussed earlier).

6.3 Parallel execution

As discussed above, iterators can be data-driven or demand-driven. The former mechanism is most suitable for single-thread query execution, whereas the latter is often considered superior in parallel, multi-thread query execution. Fortunately, the two models can be combined, e.g., in the *exchange operator* that exposes and supports demand-driven iteration in its interaction with neighboring iterators within a thread but employs data-driven data flow (with flow control or "back pressure," when required) across thread and machine boundaries [G 96].

Parallel execution of nested queries presents two principal difficulties not encountered in single-threaded execution. First, crossing a thread, process, or even machine boundary is quite expensive, in a variety of ways. Most obviously, data packets must be assembled and, usually, the operating system and thread scheduler get involved. In addition, some threads must wait for data, and during the wait time, other threads will take over the CPU and more importantly the CPU cache. The second principal difficulty is that there might be multiple consumer threads that need to bind parameters for and invoke producer threads. In other words, each producer thread must produce data for every one of the consumer threads. The alternative design, assigning a pool of producer threads to each consumer thread, leads to an explosion in the number of threads, in particular if a query plan employs multiple levels of nesting – thus, this design does not scale to complex queries.

Differently than the iterator methods that bind and unbind parameters (or more precisely rows of parameter sets) as well as return results (or result rows), some iterator method affect the entire stream and the entire iterator state. For those, namely *rewind*, *pause*, and *resume*, the consumer side must wait for all producers to reach the appropriate state and invoke the same method. In other words, the producer side can, for example, only *rewind* after all consumers have requested it. Similarly, each producer must invoke all producers to rewind. For *pause* and *resume*, producer side must be active if at least one consumer requires it, and may pause only if all consumers require it. This coordination happens in each consumer – only the last *pause* and the first *resume* are passed from the consumer half of the exchange iterator to its input iterator.

7 Summary and conclusions

In summary, executing nested queries efficiently is not as simple as it might seem at first sight. Carefully implemented and managed, asynchronous I/O can improve execution performance by an order of magnitude in single-user operation and by a substantial margin in multi-user operation. Sorting correlation values in the outer query, entirely or only opportunistically using readily available memory, can improve locality and lookup performance in the inner query. Caching results of nested queries, including the fact that a result is empty for a given set of correlation values, can be an alternative or complementary technique, possibly contributing performance improvements of another order of magnitude. Since there may be many correlation values, data flow techniques such as processing in batches must apply not only to intermediate result data but also to streams of correlation values, particularly in parallel database systems. Finally, since inner queries can be very complex in their own right, outer query, inner query, and their result may compete for memory and other resources, which therefore must be mediated for maximal efficiency and query performance.

Crucial policy issues have not been resolved here, in particular allocation of memory and threads. Further issues include policy settings for asynchronous I/O, batching, and caching, e.g., LRU policies for cached results. There is no well-understood and agreed-upon solution for those issues that is simple enough to implement and test; general enough to cover nested iteration with multiple levels and multiple branches at each level, and with memory- and CPU-intensive sort-, hash-, and bitmap operations at multiple levels; and robust enough for database users to accept without the need or desire for manual tuning. Working through those issues towards a solution that combines generality with simplicity and robustness is a challenging research problem with immediate practical application. If this paper has contributed stimulation for this research, it has fulfilled its purpose.

Acknowledgements

Barb Peters' and César Galindo-Legaria's helpful suggestions were greatly appreciated.

References

[B 97] Rudolf Bayer: The Universal B-Tree for Multidimensional Indexing: General Concepts. Proc. Worldwide Computing and Its Applications. Lecture Notes in Computer Science 1274, Springer 1997: 198-209.

[BC 81] Philip A. Bernstein, Dah-Ming W. Chiu: Using Semi-Joins to Solve Relational Queries. JACM 28(1): 25-40 (1981).

[BMG 93] José A. Blakeley, William J. McKenna, Goetz Graefe: Experiences Building the Open OODB Query Optimizer. SIGMOD Conf. 1993: 287-296.

[CL 94] Richard L. Cole, Goetz Graefe: Optimization of Dynamic Query Evaluation Plans. SIGMOD Conf. 1994: 150-160.

[DNB 93] David J. DeWitt, Jeffrey F. Naughton, Joseph Burger: Nested Loops Revisited. PDIS 1993: 230-242.

[G 93] Goetz Graefe: Query Evaluation Techniques for Large Databases. ACM Computing Surveys 25(2): 73-170 (1993).

[G 96] Goetz Graefe: Iterators, Schedulers, and Distributed-memory Parallelism. Software - Practice and Experience 26(4): 427-452 (1996).

[GBC 98] Goetz Graefe, Ross Bunker, Shaun Cooper: Hash Joins and Hash Teams in Microsoft SQL Server. VLDB Conf. 1998: 86-97.

[GLS 93] Peter Gassner, Guy M. Lohman, K. Bernhard Schiefer, Yun Wang: Query Optimization in the IBM DB2 Family. IEEE Data Eng. Bull. 16(4): 4-18 (1993).

[GW 89] Goetz Graefe, Karen Ward: Dynamic Query Evaluation Plans. SIGMOD Conf. 1989: 358-366.

[H 78] Theo Härder: Implementing a Generalized Access Path Structure for a Relational Database System. ACM TODS 3(3): 285-298 (1978).

[HCL 97] Laura M. Haas, Michael J. Carey, Miron Livny, Amit Shukla: Seeking the Truth About ad hoc Join Costs. VLDB Journal 6(3): 241-256 (1997).

[HN 96] Joseph M. Hellerstein, Jeffrey F. Naughton: Query Execution Techniques for Caching Expensive Methods. SIGMOD Conf. 1996: 423-434.

[K 80] Won Kim: A New Way to Compute the Product and Join of Relations. SIGMOD Conf. 1980: 179-187.

[K 80a] Robert P. Kooi, The Optimization of Queries in Relational Databases. Ph.D. thesis, Case Western Reserve University, 1980.

[K 82] Robert Kooi, Derek Frankforth: Query Optimization in INGRES. IEEE Data Eng. Bull. 5(3): 2-5 (1982).

[KGM 91] Tom Keller, Goetz Graefe, David Maier: Efficient Assembly of Complex Objects. SIGMOD Conf. 1991: 148-157.

[L 01] Johan Larson, Computing SQL Queries One Webpage at a Time. Ph.D. thesis, University of Wisconsin – Madison, 2001.

[LJB 95] Harry Leslie, Rohit Jain, Dave Birdsall, Hedieh Yaghmai: Efficient Search of Multi-Dimensional B-Trees. VLDB Conf. 1995: 710-719.

[M 97] Paul McJones (ed.): The 1995 SQL Reunion: People, Projects, and Politics. Digital Systems Research Center, Technical Note 1997-018, Palo Alto, CA. Also http://www.mcjones.org/System_R.

[MHW 90] C. Mohan, Donald J. Haderle, Yun Wang, Josephine M. Cheng: Single Table Access Using Multiple Indexes: Optimization, Execution, and Concurrency Control Techniques. EDBT Conf. 1990: 29-43.

[MJ 01] Bongki Moon, H. V. Jagadish, Christos Faloutsos, Joel H. Saltz: Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. IEEE TKDE 13(1): 124-141 (2001).

[OOW 93] Elizabeth J. O'Neil, Patrick E. O'Neil, Gerhard Weikum: The LRU-K Page Replacement Algorithm For Database Disk Buffering. SIGMOD Conf. 1993: 297-306.

[ÖW 92] Z. Meral Özsoyoglu, Jian Wang: A Keying Method for a Nested Relational Database Management System. ICDE 1992: 438-446.

[RM 00] Frank Ramsak, Volker Markl, Robert Fenk, Martin Zirkel, Klaus Elhardt, Rudolf Bayer: Integrating the UB-Tree into a Database System Kernel. VLDB Conf. 2000: 263-272.

[S 81] Michael Stonebraker: Operating System Support for Database Management. CACM 24(7): 412-418 (1981).

[SHP 96] Praveen Seshadri, Joseph M. Hellerstein, Hamid Pirahesh, T. Y. Cliff Leung, Raghu Ramakrishnan, Divesh Srivastava, Peter J. Stuckey, S. Sudarshan: Cost-Based Optimization for Magic: Algebra and Implementation. SIGMOD Conf. 1996: 435-446.

Konzeptbasierte Anfrageverarbeitung in Mediatorsystemen*

Kai-Uwe Sattler[1,2] Ingolf Geist[1] Rainer Habrecht[1] Eike Schallehn[1]

[1]Fakultät Informatik, Universität Magdeburg
[2]Institut für Informatik, Universität Halle
{kus|geist|habrecht|eike}@iti.cs.uni-magdeburg.de

Abstract: Ein Weg zur Überwindung der Heterogenität bei der Datenintegration in Mediatorsystemen ist die Nutzung semantischer Metadaten in Form eines Vokabulars oder einer Ontologie zur expliziten Modellierung des Hintergrundwissens. Dementsprechend muss die verwendete Anfragesprache diese Metaebene in die Anfrageformulierung und -auswertung einbeziehen. In diesem Beitrag wird hierzu ein Mediator für kulturhistorische Datenbanken vorgestellt, der auf einem konzeptbasierten Integrationsmodell basiert und die Anfragesprache CQuery – eine XQuery-Erweiterung – implementiert. Weiterhin werden ausgehend von der Semantikdefinition der Anfrageoperationen die Phasen des Rewriting, der Ausführung sowie die Nutzung eines Anfrage-Caches beschrieben.

1 Einführung

Der integrierte Zugriff auf heterogene Datenbestände stellt gerade im World Wide Web eine große Herausforderung dar. Probleme wie hohe Autonomie und Heterogenität der Quellen oder auch Skalierbarkeit und Evolutionsfähigkeit hinsichtlich einer großen Anzahl teilweise noch wechselnder Quellen treten hier im besonderen Maße zutage. Die Bandbreite möglicher Lösungsansätze reicht dabei von einfachen (Meta-)Suchmaschinen über materialisierende Ansätze bis hin zu Mediatorsystemen, die Anfragen auf einem globalen Schema in Teilanfragen zerlegen, an die Quellsysteme zur Bearbeitung weiterleiten und die Ergebnisse zusammenführen.

Mediatorsysteme lassen sich grob danach unterscheiden, wie die Korrespondenzen zwischen den lokalen Schemata der Quellen und dem globalen Mediatorschema gestaltet sind. Beim Global-as-View-Ansatz (GaV) wird das globale Schema als Sicht über den lokalen Schemata definiert. Dagegen wird beim Local-as-View-Ansatz (LaV) von einem globalen Schema ausgegangen, über das schließlich die lokalen Schemata als Sichten definiert werden, die somit nur Ausschnitte sowohl bezüglich des Schemas als auch der Extensionen beinhalten. Der GaV-Ansatz hat den Vorteil einer einfacheren Anfrageverarbeitung, beim LaV-Prinzip ist das Hinzufügen bzw. Entfernen von Quellen deutlich einfacher, da hier keine Korrespondenzen zwischen Quellen berücksichtigt werden müssen [Hal01].

*Diese Arbeit wurde teilweise gefördert von der DFG (FOR 345/1) sowie der Koordinierungsstelle für Kulturgutverluste.

Bei Mediatorsystemen der ersten Generation erfolgt die Integration im Wesentlichen auf struktureller Ebene. Die Daten aus den einzelnen Quellen werden dabei anhand struktureller Korrespondenzen wie die Zugehörigkeit zu gleich strukturierten Klassen oder die Existenz gemeinsamer Attribute kombiniert. Dies funktioniert am besten bei relativ homogenen Strukturen der beteiligten Quellen. In Szenarien mit eher disjunkten Domänen führt dies jedoch zu einer Vielzahl globaler Klassen, was wiederum detailliertes Hintergrundwissen zur Formulierung der daraus resultierenden komplexeren Anfragen erfordert.

Ein Ausweg ist die explizite Modellierung dieses Hintergrundwissen in Form semantischer Metadaten, d.h. als Vokabular, Begriffs- bzw. Konzepthierarchie oder gar Ontologie, und deren Nutzung zur Anfrageverarbeitung und Datenintegration. Ähnliche Bemühungen gibt es im Rahmen des Semantic Web, wo die (wissensbasierte) Verarbeitung von Web-Dokumenten durch das Hinzufügen einer semantischen Ebene mit Metadaten erleichtert werden soll. Erste Ergebnisse dieser Arbeiten sind Modelle und Sprachen für Vokabulare und Ontologien, wie z.B. RDF Schema (RDFS) und DAML+OIL, sowie damit verbundene Technologien. Da hierbei grundsätzliche sehr ähnliche Problemstellungen bestehen, liegt eine Verbindung von Semantic Web und Datenbank-/Mediatortechnologien nahe. Als besondere Anforderung aus Sicht der Datenintegration ergibt sich jedoch die Notwendigkeit, den Bezug zu den Quelldaten herzustellen, indem beschrieben wird, wie die Quellsysteme ein gegebenes Konzept der semantischen Ebene strukturell und inhaltlich realisieren. Diese Informationen müssen beim Registrieren einer Quelle bereitgestellt und als Teil der Anfragetransformation und -zerlegung ausgewertet werden.

Vor diesem Hintergrund wird im Folgenden der YACOB-Mediator vorgestellt, der zur Formulierung und Auswertung von Anfragen über heterogenen Web-Quellen explizit modelliertes Domänenwissen in Form von Konzepten und deren Beziehungen nutzt. Dieses System wurde für die integrierte Suche in kulturthistorischen Datenbanken entwickelt, die Informationen über kriegsbedingt verlorengegangene Kulturgüter verwalten.

Der Beitrag dieses Ansatzes liegt in *(i)* der Definition eines auf RDFS basierenden Metamodells, das die Repräsentation von Konzepten und Beziehungen als begriffliche „Anker“ für die Integration mit den Informationen zur Abbildung auf die Quelldaten verbindet und dabei dem Local-as-View-Prinzip folgt, *(ii)* der Anfragesprache CQuery, welche die Formulierung von Anfragen sowohl über die Konzept- als auch die Instanzebene unterstützt und *(iii)* der Darstellung von Operationen und Strategien zur Transformation, Zerlegung und Ausführung globaler CQuery-Anfragen inklusive eines semantischen Caching.

2 Ein konzeptbasiertes Modell zur Datenintegration

Zur Repräsentation der zu integrierenden Daten und der ihnen zuzuordnenden semantischen Metadaten sind zwei Ebenen zu berücksichtigen: *(i)* die eigentliche Daten- oder Instanzebene, welche die in den Quellsystemen gespeicherten Daten umfasst sowie *(ii)* die Meta- oder Konzeptebene zur Beschreibung der Semantik der Daten und deren Beziehungen. Als Datenmodell für die Instanzebene (im Folgenden nur als „Datenmodell“ bezeichnet) wird in unserem Ansatz ein einfaches semistrukturiertes Modell auf Basis von

XML eingesetzt. Im Mediator und im Austausch mit den Quellsystemen bzw. deren Wrappern werden Daten somit konsequent in XML repräsentiert. Dementsprechend wird davon ausgegangen, dass ein Quellsystem Daten in XML exportiert und als Anfragen einfache XPath-Ausdrücke verarbeiten kann. Eine gegebenenfalls notwendige Transformation von Anfragen beispielsweise nach SQL wird dadurch vereinfacht, dass nur eine eingeschränkte Untermenge von XPath verwendet wird, die u.a. keine Funktionsaufrufe enthält. Für den Export kann eine Quelle prinzipiell XML-Daten mit einer beliebigen DTD liefern, da die notwendigen Transformationen automatisch durchgeführt werden können.

Das Modell für die Beschreibung der Konzeptebene (im Folgenden „Konzeptmodell“ genannt) basiert auf RDF Schema. Das Resource Description Framework (RDF) – vom W3C als Mechanismus zur Erzeugung und zum Austausch von Metadaten für Web-Dokumente entwickelt – ist ein einfaches graphbasiertes Modell, dessen Knoten Ressourcen oder Literalen und dessen Kanten Eigenschaften entsprechen. RDF Schema (RDFS) definiert darauf aufbauend weitergehende Primitive wie Klassen und Klassenbeziehungen. Obwohl eine grundsätzliche Verwandtschaft mit klassischen Datenbankmodellen besteht, weist RDFS dennoch einige Besonderheiten auf, wie z.B. dass Properties (Eigenschaften) unabhängig von Klassen definiert und nur über Domain Constraints bezüglich ihrer Anwendbarkeit auf bestimmte Klassen eingeschränkt werden.

Mit Modellierungsprimitiven wie *Class*, *Property* oder *subClassOf* ermöglicht RDFS die Beschreibung einfacher Vokabulare oder Ontologien. Ein Beispiel aus dem hier betrachteten Anwendungsbereich ist in Abb. 1 dargestellt. Ein Konzept im Mediatormodell entspricht somit einer Klasse in RDFS (genauer: ein Konzept wird als Subklasse einer speziellen, hier nicht dargestellten Klasse *Concept* definiert), Konzepteigenschaften werden als Properties abgebildet. Beziehungen zwischen Konzepten, die über die bereits in RDFS definierten Beziehungen wie *subClassOf* hinausgehen, werden ebenfalls als Property modelliert, wobei als Werte- und Bildbereich solcher Properties nur Klassen zugelassen sind.

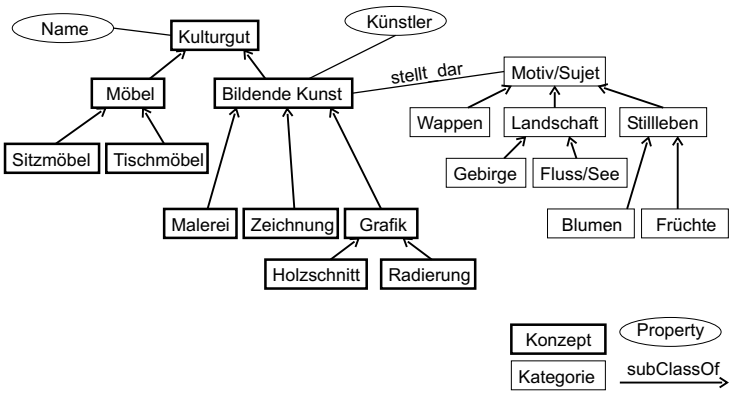


Abbildung 1: Konzepthierarchie

Die Konzeptebene übernimmt somit die Rolle eines globalen Schemas des Mediators, erlaubt aber darüber hinaus auch noch weitergehende semantische Modellierungen. So las-

sen sich nicht nur Schemainformationen in Form von Klassen bzw. Konzepten sondern auch Ausprägungen in Form von Werten darstellen. Hierzu werden Kategorien als eine spezielle Variante einer RDFS-Klasse eingeführt, die im Gegensatz zu Konzepten keine Extensionen aufweist. Eine Kategorie kann als Begriff angesehen werden, der in verschiedenen Quellen durch unterschiedliche Werte repräsentiert wird. Kategorien können unter Verwendung der *subClassOf*-Beziehung in Hierarchien organisiert werden.

Ein Beispiel für den Einsatz von Kategorien ist das Property *stellt_dar* zum Konzept „*Bildende Kunst*“ in Abb. 1. Der Wertebereich dieses Properties ist durch die Kategorie „*Motiv/Sujet*“ definiert, das wiederum verschiedene Unterkategorien besitzt. In einer Quelle könnte eine dieser Kategorien bspw. durch ein Literal „still life“ als Wert des das Property *stellt_dar* repräsentierende Element bzw. Attribut dargestellt werden, in einer zweiten Quelle durch eine beliebige andere Kodierung etc. Auf diese Weise lassen sich Informationen explizit machen, die anderenfalls in den Daten verborgen sind.

Formal lassen sich Konzept- und Datenmodell wie folgt beschreiben. Seien *URI* die Menge der Uniform Resource Identifier, *Name* eine Menge gültiger Bezeichner und $\mathcal{L}$ die Menge aller Literale. Die Menge der Klassen wird bezeichnet mit $\mathcal{T} = \mathit{URI} \times \mathit{Name}$ bestehend aus einer eindeutigen URI und einem Klassennamen. Klassen können weiter unterschieden werden in

- Konzepte (bezeichnet mit $\mathcal{C} \subset \mathcal{T}$) als Klassen zu denen Extensionen in den Quellsystemen existieren
- und Kategorien (bezeichnet mit $\mathcal{V} \subset \mathcal{T}, \mathcal{V} \cap \mathcal{C} = \emptyset$), die als abstrakte Eigenschaftwerte zur semantischen Gruppierung von Objekten verwendet werden können, jedoch keine Extension aufweisen.

Zu den Konzepten können Properties $\mathcal{P}$ definiert werden mit $\mathcal{P} = \mathit{Name} \times \mathcal{C} \times \{\mathcal{T} \cup \mathcal{L}\}$, die jeweils aus einem Bezeichner, einem Konzept, dem sie zugeordnet sind, und einer Klasse als Wertebereich bzw. einem Literal als Ausprägung bestehen. Ferner sei eine Spezialisierungsbeziehung **is_a** $\subseteq \mathcal{T} \times \mathcal{T}$ gegeben, wobei Spezialisierungshierarchien von Konzepten und Kategorien immer disjunkt sein müssen.

Das Datenmodell kann in Anlehnung an OEM [GMPQ$^+$97] beschrieben werden. Die Menge aller Objekte sei als $\mathcal{O} = \mathit{ID} \times \mathit{Name} \times \{\mathcal{L} \cup \mathit{ID} \cup \mathbb{P}\mathit{ID}\}$ definiert, wobei für ein Objekt $(\mathit{id}, \mathit{name}, v) \in \mathcal{O}$ *id* eine eindeutige Objekt-ID, *name* den Elementname und *v* den Wert des Objektes bezeichnen. *v* kann dabei ein atomarer Wert, eine Objekt-ID oder eine Menge von Objekt-IDs sein.

Die Extension **ext** : $\mathcal{C} \rightarrow \mathcal{O}$ eines Konzeptes $c = (\mathit{uri}, \mathit{name})$ umfasst eine Menge von Objekten, deren Elementname gleich dem Konzeptnamen ist und die Objekte, auf die im Rahmen der Menge *v* verwiesen wird, den zu dem Konzept definierten Properties entsprechen:

$$\mathbf{ext}(c) = \{o = (\mathit{id}, \mathit{name}, v) \mid \mathit{name} = c.\mathit{name} \wedge \forall (p, c, c') \in \mathcal{P} : \exists w \in v : w.\mathit{name} = p\}$$

Eine weitere wesentliche Komponente des Mediatormodells ist die Beschreibung der Abbildung auf die Schemata der Quellen, d.h. konkret die Angaben, wie eine Quelle Daten

zu einem gegebenen Konzept liefert und wie deren Struktur auf die durch das Konzept und dessen Eigenschaften definierte globale Struktur abgebildet wird. Hierbei werden *(i)* entsprechend der RDFS-Modellierung Konzepte und Eigenschaften getrennt betrachtet und *(ii)* das LaV-Prinzip verfolgt, d.h. ein Quellenschema wird bezüglich des globalen Schemas (hier des Konzeptschemas) definiert.

Als Konsequenz werden eine Konzept-Mapping- und eine Property-Mapping-Vorschrift benötigt, die wiederum als RDFS-Klassen realisiert sind. Eine Instanz eines Konzept-Mappings legt fest, wie das zugeordnete Konzept von einem Quellsystem unterstützt wird. Hier können zwei Fälle unterschieden werden:

(a) ein Quellsystem liefert Instanzen, die alle durch das globale Konzept definierten Eigenschaften (Attribute) unterstützen,

(b) ein Quellsystem liefert nur partielle Informationen, d.h. die Instanzen müssen durch Daten (Attributwerte) aus anderen Quellen vervollständigt werden.

Für beide Fälle umfasst ein Konzept-Mapping *CM* die Komponenten Quellenname, lokaler Elementname sowie optional ein Filterprädikat: *CM = (Source, LName, FilterPredicate)*. Über den Quellenname kann die Quelle identifiziert werden, wenn Instanzen zu diesem Konzept angefragt werden. Der lokale Elementname bezeichnet das XML-Element, dass in der Quelle zur Repräsentation von Instanzen dieses Konzeptes verwendet wird. Mit dem Filterprädikat in Form eines XPath-Ausdrucks kann die Instanzmenge eingeschränkt werden. Dies ist beispielsweise notwendig, wenn im Quellsystem verschiedene Konzepte durch die gleiche Extension (d.h. mit dem gleichen XML-Elementnamen) jedoch mit unterschiedlichen Attributwerten (z.B. `typ = 'Bild'`) dargestellt werden. Die Existenz eines Konzept-Mappings gibt somit an, dass eine Quelle eine Teilmenge der Extension des zugeordneten Konzeptes liefert.

Eine Property-Mapping-Struktur *PM* definiert die Abbildung einer Konzepteigenschaft auf Elemente bzw. Attribute der Quelldaten. Hier umfasst die Mapping-Struktur die Komponenten Quellenname sowie den Pfad zum die Eigenschaft repräsentierenden Element: *PM = (Source, PathToElement)*.

Kategorien werden in den Quellen durch einfache Werte (Literale) repräsentiert. Ein Property, das als Wertebereich eine Kategorie besitzt, kann somit in der Quelle exakt die definierten Werte dieser Kategorie sowie der davon abgeleiteten Unterkategorien annehmen. Die entsprechende Value-Mapping-Struktur *VM* umfasst daher nur den Quellennamen und den die Kategorie repräsentierenden Wert in der Quelle: *VM = (Source, Literal)*.

Mit diesen Mapping-Instanzen werden nun die Elemente des Konzeptschemas „annotiert“. Zu jedem Konzept, das von einer Quelle unterstützt wird, sind somit ein Konzept-Mapping- und die zugehörigen Property-Mapping-Beschreibungen sowie eventuell notwendige Value-Mapping-Definitionen anzugeben. Da jedoch das Konzeptschema als wesentliche Beziehung die Spezialisierung bzw. Generalisierung (*subClassOf*) verwendet, muss nicht zu jedem Konzept in einer Spezialisierungshierarchie eine entsprechende Abbildung definiert werden. Vielmehr ist dies nur für die Konzepte erforderlich, die bezüglich einer Quelle Blätter der Hierarchie darstellen. In Abb. 2 ist diese Zuordnung anhand eines Beispiels illustriert.

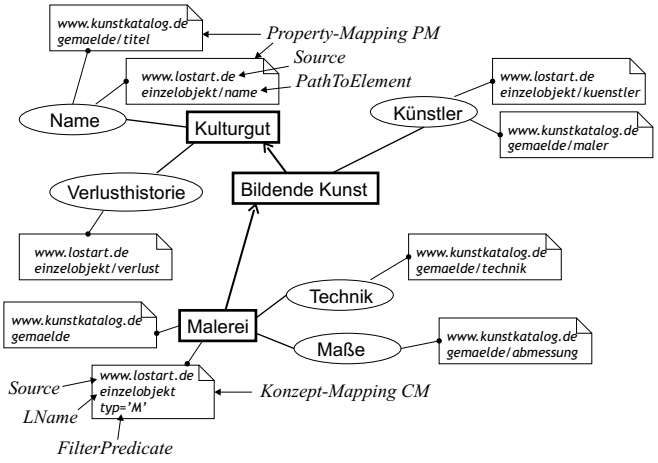


Abbildung 2: Abbildungsinformationen für die Konzepthierarchie

Das Beispiel zeigt auch, dass eine Quelle nicht notwendigerweise vollständige Daten im Sinne der Konzeptdefinition liefern muss. Werden von einer Quelle nicht alle in der Anfrage referenzierten Elemente unterstützt, so wird versucht, die fehlenden Informationen durch eine äußere Vereinigung aus einer anderen Quellen zu ergänzen. Die hierfür in Frage kommenden Quellen können anhand der vorhandenen Property-Mapping-Elemente identifiziert werden. Kann keine Quelle mit komplementären Informationen identifiziert werden, müssen im Anfrageergebnis die entsprechenden Elemente weggelassen werden (entspricht Null-Elementen).

Durch die Spezifikation der Mapping-Elemente für jede Quelle lassen sich sowohl Beschreibungskonflikte (Namenskonflikte von Elementen), strukturelle Konflikte als auch Datenkonflikte auflösen. Letztere jedoch nur für Kategorien, da hierbei eine Abbildung zwischen Werten stattfindet. Die hierfür notwendigen Transformationen werden bei der Anfrageübersetzung über die Mapping-Informationen und bei der Verarbeitung der Ergebnisse durch quellenspezifische (kompilierte) XSLT-Regeln durchgeführt, die bei der Registrierung einer Quelle automatisch generiert werden.

3 Die Anfragesprache CQuery

Das im vorangegangenen Abschnitt beschriebene Modell zur Datenintegration hat für die Planung und Ausführung von Anfragen zwei wesentliche Konsequenzen:

- Es werden Operationen benötigt, die sowohl auf Konzept- als auch auf Datenebene anwendbar sind und darüber hinaus einen Übergang zwischen beiden Ebenen ermöglichen.

- Eine globale Anfrage ist unter Verwendung der Abbildungsinformationen so zu transformieren und zu zerlegen, dass die zu den gewünschten Konzepten relevanten Quellen einbezogen und die einzelnen Teilanfragen autonom von den jeweiligen Quellen beantwortet werden können.

Zur Formulierung von Anfragen wird in dem hier vorgestellten Mediatorsystem eine CQuery genannte Variante von XQuery eingesetzt. Die Besonderheiten von CQuery sind im Wesentlichen semantischer Natur, syntaktisch folgt CQuery der FLWR-Notation von XQuery. Eine CQuery-Anfrage umfasst folgende Komponenten: die Auswahl von Konzepten anhand bestimmter Eigenschaften oder durch Operationen wie Traversierung von Beziehungen und Mengenoperationen, die Filterung der Daten als Instanzen der Konzepte sowie die Verknüpfung bzw. Projektion der Ergebnisse.

Der Aufbau einer typischen CQuery-Anfrage ist wie folgt:

Q_1 :
```
FOR $c IN concept[name='Malerei']
LET $e := extension($c)
WHERE $e/kuenstler = 'van Gogh'
RETURN
  <painting>
    <title>$e/name</title>
    <artist>$e/kuenstler</artist>
  </painting>
```

Diese Anfrage liefert ein XML-Dokument mit Informationen über Gemälde in Form von `painting`-Elementen, die wiederum aus dem Titel und dem Künstlernamen bestehen.

In CQuery-Anfragen dient die **`FOR`**-Klausel zur Auswahl. Das Pseudoelement `concept` wird dabei als Dokumentbaum aus allen definierten Konzepten interpretiert, wobei die Konzepteigenschaften ebenfalls als Element angesehen werden. Auf diese Weise lassen sich die Standardmechanismen von XQuery auch für die Anfrageteile auf Konzeptebene nutzen. Spezielle Sprachkonstrukte oder -erweiterungen, wie sie etwa für RDF-Anfragesprachen vorgeschlagen wurden, sind somit nicht notwendig. Neben Selektionen bezüglich Konzepteigenschaften lassen sich auch Mengenoperationen wie **`UNION`**, **`EXCEPT`** und **`INTERSECT`** zwischen Konzeptmengen sowie Traversierung von Konzeptbeziehungen nutzen. Beziehungen werden dabei ebenfalls als Elemente aufgefasst, so dass für das Konzeptschema aus Abb. 1 der folgende Ausdruck

```
concept[name='Bildende Kunst']/!subClassOf
```

die direkt vom Konzept „Bildende Kunst“ abgeleiteten Konzepte liefert. Das dem Beziehungsnamen vorangestellte „!“-Zeichen gibt dabei an, dass die zu `subClassOf` inverse Beziehung (die hier nicht explizit benannt ist) betrachtet werden soll. Ein dem Beziehungsnamen folgendes „+“-Zeichen erzwingt dagegen die Bestimmung der transitiven Hülle bezüglich dieser Beziehung. So würde

```
concept[name='Bildende Kunst']/!subClassOf+
```

alle direkt oder indirekt vom Konzept „Bildende Kunst“ abgeleiteten Konzepte liefern. Zu beachten ist jedoch, dass speziell für die `subClassOf`-Beziehung die explizite Angabe

von „+“ nicht erforderlich ist, da bei der Bestimmung der relevanten Quellen die Transitivität dieser Beziehung berücksichtigt wird. Dies bedeutet, dass für die obige CQuery-Anfrage nicht nur die Quellen ausgewählt werden, die exakt das Konzept „Bildende Kunst“ unterstützen, sondern alle Quellen, die ein von diesem Konzept abgeleitetes Konzept repräsentieren. Diese implizite Ermittlung der transitiven Hülle eines Konzeptes erfolgt grundsätzlich nach einer eventuell angegebenen Pfadtraversierung aber vor einer Anwendung von Mengenoperation. Somit liefert der folgende Ausdruck alle Konzepte, die Kulturgüter – jedoch keine Möbel – sind:

```
concept[name='Kulturgut'] EXCEPT concept[name='Möbel']
```

In der **LET**-Klausel wird der Übergang von der Konzept- zur Instanzebene vollzogen. Hierzu dient die vordefinierte Funktion `extension()`, die zu einem Konzept die Extension, d.h. die Menge aller Instanzen der zugeordneten Quellen, liefert. Da die Konzepte in der **FOR**-Klausel einzeln einer Variablen zugewiesen werden, wird diese Funktion für jedes Konzept ausgewertet. Auf das Ergebnis – die Instanzmenge, die wiederum an eine Variablen gebunden werden kann – kann schließlich die Filterbedingung der **WHERE**-Klausel angewendet werden. Werden der Variablen in der **FOR**-Klausel nacheinander mehrere Konzepte zugewiesen, so werden die einzelnen mit `extension()` bestimmten Instanzmengen vereinigt – diese Operation wird im weiteren als extensionaler Union bezeichnet.

Während die Operationen der **FOR**-Klausel ausschließlich auf den Metadaten, d.h. im Mediator, ausgeführt werden, sind die Auswertung der `extension()`-Funktion sowie der Filterbedingung in der **WHERE**-Klausel mit Zugriffen auf die Quellsysteme verbunden. Die `extension()`-Funktion initiiert die Ausführung der Anfrage im Quellsystem (siehe unten), wobei – sofern möglich – die Filterbedingung ebenfalls als Teil dieser Quellenanfrage übernommen wird. Verbundoperationen werden formuliert, indem in der **LET**-Klausel mehrere verschiedene Variablen für Instanzmengen verwendet werden und im **WHERE**-Teil eine entsprechende Bedingung spezifiziert wird.

Im **LET**-Teil einer Anfrage lässt sich nicht nur die Extension zu einem Konzept bestimmen. Auch Anfragen über Eigenschaften sind möglich, indem das Pseudoelement `property` als Kindelement eines Konzeptes genutzt wird, das die Menge aller zu diesem Konzept definierten Eigenschaften repräsentiert. Werden die ausgewählten Eigenschaften an eine Variable gebunden, so kann diese auch zur Selektion auf Instanzebene genutzt werden, was wiederum einer disjunktiv verknüpften Anfrage über alle definierten Eigenschaften entspricht:

Q_2 :
```
FOR $c IN concept[name='Malerei']
LET $e := extension($c), $p := $c/properties
WHERE $e/$p = 'Blumen'
  ...
```

Anfrage Q_2 liefert daher alle Gemälde, bei denen der Begriff „Blumen“ als Wert eines Properties auftaucht. Auf diese Weise sind Anfragen mit Schemaoperationen möglich, d.h. Operationen über Metadaten – in diesem Fall Informationen über Properties. Solche Operationen haben sich gerade in Anfragesprachen für heterogene Datenbanken als sehr nützlich zur Überwindung struktureller Heterogenitäten erwiesen.

In ähnlicher Form kann auf die einem Property zugeordneten Kategorien zugegriffen werden, indem im **LET**-Teil eine Variable an einen Pfadausdruck mit einem entsprechenden Property gebunden wird. Diese Variable verweist danach auf eine Menge von Kategorien und kann im **WHERE**-Teil anstelle eines Wertes eingesetzt werden:

Q_3 :
```
FOR $c IN concept[name='Malerei']
LET $e := extension($c),
    $k := $c/stellt_dar[name='Stillleben']
WHERE $e/stellt_dar = $k
  ...
```

Mit dieser Anfrage wird zunächst der Variablen `$k` die Menge der Kategorien von Stillleben zugewiesen, indem die Menge aller Kategorien zur Beziehung `stellt_dar` bestimmt und diese dann auf die „Stillleben“ reduziert wird. Dies schließt auch alle von der konkreten Kategorie „Stillleben“ abgeleiteten Kategorien ein. In die Bedingung im **WHERE**-Teil wird dann der für die jeweilige Quelle lokal verwendet Begriff für „Stillleben“ bzw. im Fall von mehreren Kategorien eine disjunktive Verknüpfung über alle übersetzten Begriffe eingesetzt. Das Ergebnis der Anfrage Q_3 umfasst somit alle Gemälde, die ein Stillleben darstellen.

Die **RETURN**-Klausel hat die gleiche Bedeutung wie in XQuery: Hier kann das Anfrageergebnis entsprechend einer vorgegebenen XML-Dokumentstruktur ausgegeben werden, wobei die Ergebniselemente (d.h. sowohl die Instanzen als auch die Konzepte) über die eingeführten Variablen referenziert werden.

4 Anfragebearbeitung

Die Semantik der Anfragesprache lässt sich auf einfache Weise mit Hilfe von Algebraoperationen beschreiben. Ausgehend von den Definitionen aus Abschnitt 2 können zwei Gruppen von Anfrageoperationen unterschieden werden, die im Wesentlichen den bekannten Operationen der Relationenalgebra entsprechen. Operationen der Konzeptebene sind die Selektion Σ_{Cond}, die bekannten Mengenoperationen $\cup$, $\cap$, $-$ sowie die Pfadtraversierung $\Phi_P(c)$, die zu jedem Konzept einer Menge C alle über die Beziehung p referenzierten Klassen liefert:

$$\Phi_p(C) = \{c' \mid \exists c \in C : (p, c, c') \in \mathcal{P}\}$$

Die Traversierung einer inversen Beziehung kann entsprechend formuliert werden:

$$\Phi_{\overline{p}}(C) = \{c' \mid \exists c \in C : (p, c', c) \in \mathcal{P}\}$$

Weiterhin wird für die Berechnung der transitiven Hülle eines Konzeptes (bzw. einer Konzeptmenge) bezüglich einer Beziehung p eine eigene Operation Φ_p^+ eingeführt:

$$\Phi_p^+(C) = \{c' \mid \exists c_s \in C : (p, c_s, c') \in \mathcal{P} \vee \exists c_i \in \Phi_p^+(\{c_s\}) : (p, c_i, c') \in \mathcal{P}\}$$

Die Operationen der Instanzebene sind Selektion (σ), Projektion (π) sowie das kartesische Produkt ($\times$). Eine in CQuery formulierte Anfrage kann nun wie folgt in einen Ausdruck aus den obigen Operationen übersetzt werden.

1. Die Klausel **FOR** `$c` **IN** `concept[`*Cond*`]` wird in einen Ausdruck der Form $\Phi^{+}_{\overline{\textbf{is\_a}}}(\Sigma_{Cond}(\mathcal{C}))$ überführt. Hierbei steht $\overline{\textbf{is\_a}}$ für die inverse Beziehung zu c_2 **is_a** c_1.

2. Traversierungen der Form `concept[`*Cond*`]/`*prop*$_1$`/.../`*prop*$_n$ werden in $\Phi^{+}_{\textbf{is\_a}}(\Phi_{prop_n}(\ldots\Phi_{prop_1}(\Sigma_{Cond}(\mathcal{C}))))$ übersetzt.

3. Die Anwendung von Mengenoperationen wie `concept[`*Cond*$_1$`]` **UNION** `concept[`*Cond*$_2$`]` liefert folgenden Ausdruck:

$$\Phi^{+}_{\textbf{is\_a}}(\Sigma_{Cond_1}(\mathcal{C})) \cup \Phi^{+}_{\textbf{is\_a}}(\Sigma_{Cond_2}(\mathcal{C}))$$

4. Ein Ausdruck **LET** `$e := extension($c)` **WHERE** *Cond* wird transformiert, indem die Übersetzung *CExpr* des an die Variablen `$c` gebundenen Ausdrucks in folgenden Ausdruck eingesetzt wird, wobei $\biguplus$ für die äußere Vereinigung steht:

$$\biguplus_{c \in CExpr} \sigma_{Cond}(\textbf{ext}(c))$$

   Die Verwendung der äußeren Vereinigung als Integrationsoperation erlaubt die Kombination von Datenquellen mit teilweiser Überlappung bzw. mit einer partiellen Union-Kompatibilität der Schemata, wobei angenommen wird, dass die in allen Relationen vorhandenen Attribute (bzw. Elemente) Schlüsseleigenschaften aufweisen. Tritt ein Tupel mit seinem Schlüsselwert in allen Eingangsrelationen auf, so wird es nur einmal in die Ergebnismenge aufgenommen. Anderenfalls werden die fehlenden Attribute bzw. Elemente weggelassen, was dem Auffüllen mit Nullwerten im relationalen Fall entspricht.

5. Werden in der **LET**-Klausel mehrere Variablen für Instanzmengen bzw. Property- oder Kategoriemengen eingeführt, so wird das kartesische Produkt über diesen Mengen gebildet.

6. Die **RETURN**-Klausel wird in eine entsprechende Projektion π umgesetzt, die neben XML-Tags als Literale auch Pfadausdrücke zur Projektion von Attributen umfasst.

Für eine Anfrage

Q_5 :
```
FOR $c IN concept[name='Malerei']
LET $e := extension($c)
WHERE $e/kuenstler = 'van Gogh'
RETURN
  <painting>
    <title>$e/title</title>
    <artist>$e/kuensler</artist>
  </painting>
```

ergibt sich unter Anwendung der obigen Regeln folgender Algebraausdruck, wobei *Proj* für den Projektionsausdruck aus Q_5 steht:

$$\biguplus_{c \in \Phi^{+}_{\textbf{is\_a}}(\Sigma_{\text{name='Malerei'}}(\mathcal{C}))} \pi_{Proj}(\sigma_{\text{kuenstler='van Gogh'}}(\textbf{ext}(c)))$$

Die Übersetzung einer CQuery-Anfrage in die interne Algebraform bildet auch gleichzeitig die Vorbereitung für die eigentliche Anfragebearbeitung, deren Ablauf in Abb. 3 dargestellt ist.

Gegeben:
Anfrageausdruck der Form $\biguplus_{c \in CExpr} IExpr(c)$
Ergebnis $R := \{\}$

Berechne Konzeptmenge $C :=$ *CExpr*
forall $c \in C$ **do**
/ Anfrage im Cache suchen → Ergebnis ist R_c */*
$R_c := \text{cache-lookup}(IExpr(c))$
if $R_c \neq \{\}$ **then**
/ Cache-Eintrag gefunden */*
$R := R \uplus R_c$
else
/ Quellenanfrage ableiten */*
Bestimme alle in *IExpr* referenzierten Properties p_i und Kategorien k_j
forall CM_s zu c **do**
$s :=$*CM*$_s(c)$*.Source*
/ nur nicht-redundante Konzepte anfragen */*
if $c \in \text{cmin}(C, s)$ **then**
/ zu jeder unterstützenden Quelle eine eigene Anfrage konstruieren */*
Transformiere *IExpr(c)* entsprechend $CM_s(c)$, $PM_s(p_i)$ und $VM_s(k_j)$ in eine Quellenanfrage Q
Führe Q an Quelle $CM_s(c)$*.Source* aus; Ergebnis ist R_Q
$R := R \uplus R_Q$
fi
od
fi
od

Abbildung 3: Ablauf der Anfragebearbeitung

Dieser Algorithmus verarbeitet nur elementare Anfrageausdrücke über Konzeptmengen und deren Extensionen. Anfragen mit kartesischen Produkt bzw. Verbund werden in elementare Teilausdrücke zerlegt, die dann in der dargestellten Weise verarbeitet werden können. Dabei wird auch versucht, durch einfache Heuristiken eine Vereinfachung der Anfragen vorzunehmen. Hierzu zählen neben den bekannten algebraischen Optimierungsregeln („Herunterdrücken“ von Selektionen) insbesondere die Ersetzung von Zugriffen auf Schemaelemente (wie in Anfrage Q_2) durch disjunktiv verknüpfte Bedingungen sowie die Ersetzung von Kategorievariablen durch die entsprechenden Wertemengen (Anfrage Q_3).

Der erste Schritt ist die Auswertung der Operationen auf Konzeptebene. Zu jedem der dadurch ermittelten Konzepte wird zunächst versucht, den extensionsbezogenen Teil *IExpr* der Anfrage aus dem Cache zu beantworten (siehe hierzu Abschnitt 5). Schlägt dies

fehl, werden alle Konzept-Mappings und damit alle dieses Konzept unterstützende Quellen bestimmt. Da zwischen den Extensionen verschiedener Konzepte einer Quelle Überlappungen bzw. Redundanzen existieren können, werden nur nicht-redundante Konzepte berücksichtigt. Diese werden mit Hilfe der Funktion $\mathrm{cmin}(C, s)$ ermittelt, die zu einer gegebenen Quelle s die minimale Teilmenge der Konzeptmenge C mit Hilfe folgender Heuristiken bestimmt.

Eliminierung Betrachtet man Konzepthierarchien als Klassenhierarchien mit extensionalen Beziehungen, so könnte bei einer Beziehung c_2 **is_a** c_1 zwischen zwei Konzepten c_1 und c_2, die von der selben Quelle unterstützt werden, die Anfrage zur Bestimmung der Extension von c_2 entfallen, da **ext**$(c_2) \subseteq$ **ext**(c_1) gilt. Da in dem hier betrachteten Szenario Konzepthierarchien nicht als Ergebnis einer Schemaintegration definiert werden, gilt diese Annahme jedoch nicht zwingend, sondern nur dann, wenn beide Konzepte in der Quelle durch die gleiche Klasse, d.h. repräsentiert durch das gleiche lokale Element im Konzept-Mapping *CM*(c), unterstützt werden:

$$c_2 \textbf{ is\_a } c_1 \wedge \mathit{CM}(c_1).\mathit{LName} = \mathit{CM}(c_2).\mathit{LName} \Rightarrow \textbf{ext}(c_2) \subseteq \textbf{ext}(c_1)$$

Zusammenfassen Teilanfragen zu Konzepten in parallelen Zweigen einer Konzepthierarchie, die sich auf die gleiche Quelle beziehen, können zusammengefasst werden, wenn sie auch die gleiche Klasse betreffen, d.h. wenn $\mathit{CM}(c_1).\mathit{LName} = \mathit{CM}(c_2).\mathit{LName}$. In diesem Fall können die gegebenenfalls spezifizierten Filterbedingungen der Konzept-Mappings disjunktiv verknüpft werden.

Im nächsten Schritt werden die verbliebenen Teilanfragen anhand der Mappings in Quellenanfragen übersetzt. Das Konzept-Mapping liefert dabei den Bezeichner des lokalen Elementes, die Property-Mappings zu den dem Konzept zugeordneten Properties die Bezeichner der Attribute und die Value-Mappings die Übersetzung von Begriffen. Eine Übersetzung des Anfrageausdrucks $\sigma_{P\theta v}(\textbf{ext}(c))$ wird demnach unter Verwendung der Mapping-Informationen *CM*(c) und *PM*(p) in einen XPath-Ausdruck der Form

$$/\langle \mathit{CM}(\mathrm{c}).\mathit{LName}\rangle[\langle \mathit{PM}(\mathrm{p}).\mathit{PathToElement}\rangle\ \theta\ \mathrm{v}]$$

überführt. Quellenanfragen sind demzufolge nur einfache Selektionen über lokalen Extensionen, die auch von Nicht-DBMS-Quellsystemen beantwortet werden können. Darüber hinausgehende Operationen wie Vereinigung oder Verbund werden ausschließlich vom Mediator ausgeführt. Ein Delegieren dieser Operationen an die Quellsysteme würde die Berücksichtigung von Quelleneigenschaften erforderlich machen, wie dies u.a. in [RS97] diskutiert wird.

5 Caching von Anfrageergebnissen

Bei der virtuellen Integration von Datenbeständen ergeben sich hohe Anforderungen an die Anfragebearbeitung, um global eine effiziente Ausführung zu gewährleisten. Dies trifft insbesondere zu, wenn die Integration über Datennetze mit begrenztem Durchsatz wie dem Web geschieht. Dabei umfasst die Effizienz einerseits, dem Nutzer des globalen Systems in

angemessenen Antwortzeiten Ergebnisse bereitzustellen, und andererseits die Auslastung der integrierten Quellen durch redundante oder irrelevante Anfragen zu minimieren.

Das semantische Caching von Anfragen und zugehörigen Ergebnissen hat sich in vergleichbaren Szenarien als geeignetes Mittel zur Erfüllung der genannten Anforderungen erwiesen. Dabei wird auf der Basis zuvor zwischengespeicherter Anfragen entschieden, ob eine neue Anfrage ganz oder teilweise aus den zugehörigen zwischengespeicherten Anfrageergebnissen beantwortet werden kann. Für die Umsetzung eines solchen semantischen Caches sind zu berücksichtigen *(i)* die Integration in die globale Anfrageverarbeitung, *(ii)* die Anbindung an das Konzeptmodell, *(iii)* die physische Speicherung der Anfragen und Ergebnisse sowie *(iv)* eine Cache-Management-Strategie.

Da im YACOB-System der Schwerpunkt des Caching auf der Unterstützung einer schrittweisen interaktiven Anfrageformulierung liegt, betrachten wir im Folgenden vor allem folgende Fälle der Anfrageverfeinerung:

1. Einschränkung der Ergebnismenge durch die Verminderung der Menge relevanter Konzepte bzw. die Verminderung der Menge relevanter Instanzen durch Angabe zusätzlicher konjunktiv verknüpfter Prädikate oder die Entfernung disjunktiv verknüpfter Prädikate.

2. Erweiterung der Ergebnismenge durch die Erweiterung der Menge relevanter Konzepte bzw. die Erweiterung der Menge relevanter Instanzen durch die Entfernung konjunktiv verknüpfter Prädikate oder die zusätzliche Angabe disjunktiv verknüpfter Prädikate.

Die angegebenen Fälle korrespondieren zu der in Abschnitt 3 beschriebenen Trennung der Konzept- und der Instanzebene entsprechend der **LET**-Klausel. Durch ein Caching unterhalb der Konzeptebene im Mediator kann die Berücksichtigung zwischengespeicherter Inhalte auf die Untersuchung der Filterprädikate entsprechend der **WHERE**-Klausel reduziert werden. Das heißt, zu jedem Konzept existiert somit ein Cache-Fragment, in dem Paare bestehend aus einer Anfrage und den dazugehörigen transformierten Ergebnismengen bestehen. Dabei bestehen die Anfragen an dieser Stelle nur aus einfachen Selektionsbedingungen konjunktiv verknüpfter Konstantenselektionen. Disjunktiv verknüpfte Prädikate erhalten jeweils einen eigenen Cache-Eintrag.

Ein Cache-Eintrag besteht jeweils aus der oben beschriebenen *condition* und dem zugehörigen Ergebnis *instances*. Davon und vom Algorithmus aus Abb. 3 ausgehend wird der Cache entsprechend dem Algorithmus in Abb. 4 ausgewertet. Dazu wird zuerst die Filterbedingung in konjunktive Normalform zerlegt, d.h. eine Menge von Konjunktionen entsprechend der Beschreibung von Cache-Einträgen im *condition*-Teil gebildet. Die Konjunktionen, bestehend aus einer Menge von Prädikaten, werden einzeln mit den Cache-Einträgen verglichen. Bestehen beide aus gleichen Prädikaten, mit gleichem Attributnamen, Operator und Vergleichswert, repräsentieren die zwischengespeicherten Instanzen das zugehörige Ergebnis. Ist die Anfragekonjunktion spezieller, d.h. sie umfasst zusätzliche Prädikate, wird die Teilanfrage in den zwischengespeicherten Instanzen ausgewertet. Ein weiterführender Algorithmus könnte bei Überlappung der Prädikatmengen oder der von den Prädikaten beschriebenen Datenbereiche eine komplementäre Anfrage bilden, wozu jedoch

Gegeben:
Filterausdruck *IExpr*(c) über Konzept c
Cache P_c für Konzept c
Ergebnis $R := \{\}$

cache-lookup(*IExpr*(c)):
DNF := disjunctive-normalform(*IExpr*(c))
forall *conj* $\in$ *DNF* **do**
forall *ce* $\in P_c$ **do**
if *ce.condition* $=$ *conj* **then**
/ Cache-Eintrag ist Ergebnis der Teilanfrage */*
$R := R \cup$ *ce.instances*
else if *ce.condition* $\subset$ *conj* **then**
/ Teilanfrage kann aus Cache-Eintrag beantwortet werden */*
Führe die *conj* entsprechende Anfrage über die
Cache-Daten in *ce.instances* aus, Ergebnis ist R_{conj}
$R := R \cup R_{conj}$
fi
od
od

Abbildung 4: Auswertung des Cache während der Anfrageverarbeitung

eine Auswertung der Prädikatsemantik erforderlich wäre. Das beschriebene Vorgehen ist für die betrachtete Anwendung jedoch ausreichend, da die Prädikate aufgrund der Charakteristika der Daten hauptsächlich Tests auf Zeichenketten-Gleichheit enthalten.

Die physische Speicherung der Anfragen und Dokumentsegmente erfolgt mit Hilfe der XML-Datenbank Xindice. Neben Vorteilen wie Ausfallsicherheit und besserer Skalierbarkeit, die durch eine derartige persistente Verwaltung der Anfrageergebnisse gewährleistet wird, basiert der oben beschriebene Algorithmus zur Auswertung des Cache auf der integrierten XPath-Anfragekomponente von Xindice. Diese wird benutzt, um Teilanfragen aus Cache-Segmenten zu beantworten, wenn das korrekte Ergebnis einer spezielleren Anfrage im Segment enthalten ist.

Das Cache Management ist in diesem Szenario denkbar einfach, da der Cache hauptsächlich zur Unterstützung interaktiver Sitzungen vorgesehen ist. Die Verdrängung von Cache-Segmenten ist daher mit einem einfachen Zeitstempel an den Kopfelementen umgesetzt, der bei einem Zugriff aktualisiert wird. Überschreitet der Zeitstempel ein Vielfaches der durchschnittlichen Sitzungszeit, wird der Eintrag entfernt.

6 Architektur und Implementierung

Das YACOB-Mediatorsystem ist vollständig in Java implementiert, wobei auf eine Reihe von Standardtechnologien und frei verfügbaren Modulen zurückgegriffen wurde. So er-

folgt die Verarbeitung von XML-Daten (Parsing, XSLT-Transformation) mit JAXP (Java API for XML Processing) und der Zugriff auf die Quellsysteme bzw. Wrapper über Web Services. Zur Verwaltung und zur Manipulation des auf RDFS basierenden Konzeptmodells wird Jena – das Java-API for RDF – eingesetzt. Dieses Paket stellt nicht nur einen RDF-Parser und die entsprechenden Zugriffsschnittstellen bereit, sondern unterstützt auch die RDF-Anfragesprache RDQL sowie die persistente Speicherung von RDF-Modellen in einer relationalen Datenbank. Als Cache wird das XML-Datenbanksystem Xindice genutzt, die Benutzerschnittstelle ist über Java Server Pages realisiert.

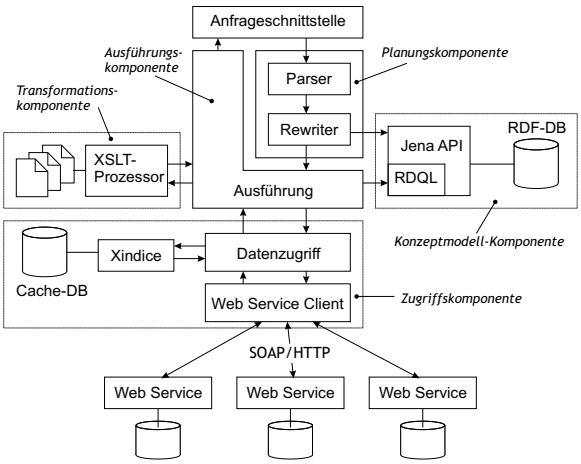


Abbildung 5: Architektur des Mediatorsystems

Das Zusammenwirken dieser Module sowie die Verbindung mit dem eigentlichen Anfragesystem ist in Abb. 5 dargestellt. Es lassen sich grob die folgenden Komponenten unterscheiden:

- die Konzeptmodell-Komponente, die auf Basis des Jena-API die Verwaltung von Konzepten, deren Eigenschaften und Beziehungen sowie der Abbildungsinformationen auf die Strukturen der Quellsysteme realisiert,
- die Anfrageplanungskomponente bestehend aus dem CQuery-Parser und dem Modul zur Anfragetransformation, -zerlegung und -übersetzung,
- die Anfrageausführungskomponente mit der Implementierung der Anfrageoperatoren sowie der Überwachung und Steuerung der Ausführung,
- die Transformationskomponente, die im Wesentlichen durch den mit JAXP bereitgestellten XSLT-Prozessor realisiert wird und die Transformation der Ergebnisdaten aus den Quellsystemen vornimmt, sowie
- die Zugriffskomponente, die XPath-Anfragen von der Ausführungskomponente entgegennimmt und entweder über das Web-Service-Protokoll SOAP an die Quellsys-

teme bzw. spezifische Wrapper weitergibt oder mit Hilfe von Xindice aus dem Cache beantwortet.

Eine Besonderheit des Systems ist die enge Verzahnung von Anfrageplanung und -ausführung. Da erst nach Ermittlung der Konzepte und deren Abbildung auf die Quellstrukturen festgestellt werden kann, welche Quellen anzufragen sind und wie die Anfragen zu übersetzen sind, muss zunächst der **FOR**-Teil einer Anfrage ausgeführt werden. Die entsprechenden Operatoren sind als Teil der Ausführungskomponente implementiert und greifen über das Jena-API und die RDF-Anfragesprache RDQL auf die RDF-Daten des Konzeptmodells zu. RDQL wird zur teilweisen Realisierung der Algebraoperatoren der Konzeptebene genutzt. Aufgrund der bestehenden Restriktionen der Sprache sind jedoch einzelne Operatoren wie die Berechnung der transitiven Hülle durch direkten Zugriff auf den RDF-Graphen implementiert. Das Ergebnis dieser Ausführung – eine Menge von Konzepten – wird an die Planungskomponente zurückgegeben, die wiederum die Mapping-Informationen über Jena bzw. RDQL ermittelt und davon ausgehend die Zerlegung und Übersetzung der Teilanfragen entsprechend den lokalen Schemata vornimmt. Dieser Plan – ein Graph aus Algebraoperatoren – wird schließlich wieder an die Ausführungskomponente übergeben, die wiederum zur Verarbeitung der entfernten Anfragen die Dienste der Zugriffskomponente in Anspruch nimmt.

XML-Daten, die als Ergebnis der Ausführung einer entfernten Anfrage von der Zugriffskomponente geliefert werden, müssen zunächst durch Anwendung der quellenspezifischen XSLT-Regeln in das globale, durch das Konzeptschema vorgegebene Schema transformiert werden. Parallel zur Weiterverarbeitung durch weitere eventuell in der Anfrage formulierte globale Operationen (z.B. Verbund, Projektion usw.) werden die transformierten Daten in der Cache-Datenbank gespeichert. Gleichzeitig werden die Mapping-Informationen zum entsprechenden Konzept mit dem Cache-Eintrag aktualisiert.

Der Zugriff auf die Quellsysteme bzw. gegebenenfalls notwendige Wrapper erfolgt über Web Services. Hierzu muss jede beteiligte Quelle eine Anfrageschnittstelle in Form eines einfachen Web Services unterstützen, der einen XPath-Ausdruck als Parameter erwartet und ein XML-Dokument mit den Anfrageergebnissen zurückliefert. Auf Mediatorseite wird der Zugriff über das Java API for XML Messaging (JAXM) realisiert, das eine einfache Generierung und Verarbeitung von SOAP-Nachrichten ermöglicht.

7 Benutzerschnittstelle

Für den Mediator wurde eine grafische Benutzerschnittstelle entwickelt, die berücksichtigt, dass die Anwender in dem betrachteten Szenario eher Kunsthistoriker, Anwälte, Vertreter von Auktionshäuser sowie andere Interessierte sind und somit mit der direkten Formulierung von Anfragen in CQuery sicher überfordert wären. Daher sollten die Informationen über Konzepte, deren Eigenschaften und Kategorien explizit einbezogen werden. Die Benutzungsoberfläche besteht dabei aus drei Teilen: die Darstellung der Konzepte als einen Baum, die Darstellung der Eigenschaften sowie die Repräsentation der Ergebnisse.

Im ersten Teil wird das RDF-Modell ausgehend von dem Wurzelkonzept „Kulturgut“

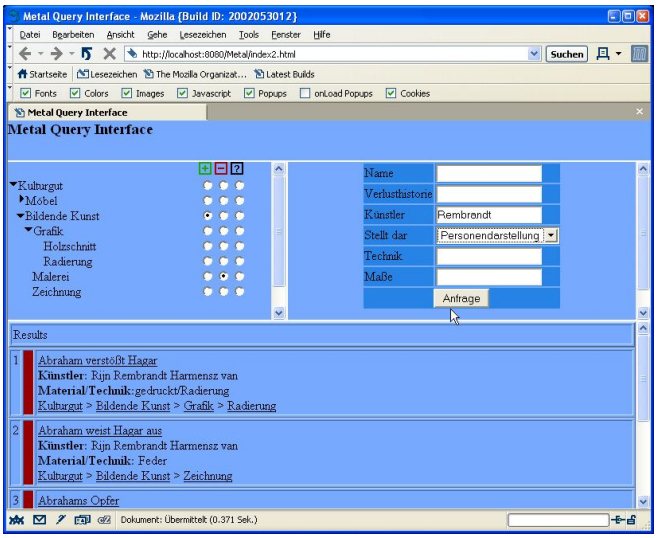


Abbildung 6: Anfragesitzung mit der Suchschnittstelle

durchlaufen und die Konzepte in einer Baumstruktur dargestellt (siehe Abb. 6). Der Nutzer kann dabei für jedes Konzept festlegen, ob Objekte des Konzeptes im Ergebnis enthalten sein dürfen, sein müssen bzw. nicht enthalten sein dürfen. Diese Informationen werden auf die **FOR**-Klausel innerhalb einer CQuery-Anfrage abgebildet.

Der zweite Teil der Oberfläche unterstützt die Behandlung der Eigenschaften zur weiteren Verfeinerung der Suche. Die Eigenschaften werden schrittweise durch die Auswahl von weiteren Konzepten ergänzt, wobei das Suchformular entsprechend angepasst wird. Hierbei sind Eigenschaften mit und ohne unterstützenden Kategorien, wie z.B. Motiv oder Epoche zu unterscheiden. Im Fall ohne Kategorieelemente wird dem Nutzer ein einfaches Texteingabefeld zur Freitextsuche angeboten. Des Weiteren werden die eingegebenen Werte in die **WHERE**-Klausel übernommen. Im zweiten Fall wird dem Nutzer eine Auswahl der verschiedenen Kategorien dargestellt. Für eine Anfrage müssen sowohl die **LET**- als auch die **WHERE**-Klausel benutzt werden.

Der dritte Teil der Benutzeroberfläche dient der Präsentation der Ergebnisse und der Verfeinerung der Anfrage. Die Ergebnisliste umfassen neben den Informationen über die Eigenschaften der Kulturgüter auch zusätzliche Angaben über die Zugehörigkeit zu den Extensionen der Konzepte. Die Zugehörigkeit wird durch einen Pfadausdruck vom Wurzelkonzept „Kulturgut“ bis zum eigentlichen Konzept ausgedrückt. Durch eine Auswahl des Konzeptes ist es möglich, die Anfrage direkt einzuschränken bzw. um zusätzliche Objekte zu erweitern. Dieses hat den Vorteil, dass dem Anwender ermöglicht wird, zusätzliche Konzepte zur Anfrage hinzufügen, sobald er ähnliche Objekte zu seinem geforderten Suchergebnis entdeckt. Diese Art der iterativen Anfrageverfeinerung profitiert besonders stark von den vorgestellten Caching-Möglichkeiten des Mediators (siehe Abschnitt 5).

8 Verwandte Arbeiten

Systeme auf der Basis der Mediator-Wrapper-Architektur haben sich in den letzten Jahren als ein geeigneter Ansatz zur Integration von heterogenen, semistrukturierten Datenquellen im Web etabliert. Als Vorreiter auf diesem Gebiet können die bekannten TSIMMIS [GMPQ+97] und Information Manifold [LRO96] angesehen werden. Aktuelle Arbeiten wie etwa MIX [BGL+99] nutzen auch bereits XML als Austauschformat bzw. XML-basierte Anfragesprachen.

Ein Vertreter der semantischen oder ontologiebasierten Integration ist das System KIND [LGM01], das sogenannte Domain Maps verwendet. Eine Domain Map wird im generischen konzeptuellen Modell GCM spezifiziert, das wiederum auf einer Teilmenge von F-Logik basiert, und als eine Menge von Klassen (Konzepten) und deren Verbindungen über binäre Relationen modelliert. Eine neue Quelle wird in KIND registriert, indem die Objekte den Konzepten der Domain Map als Instanzen zugeordnet („verankert“) werden. Über den so registrierten Quelldaten lassen sich integrierende Sichten in Form von Anfragen (Regeln) definieren. Grundsätzlich verfolgt KIND somit einen Global-as-View-Ansatz.

Ein weiterer Vertreter ist SIMS [ACHK93], das auf der Wissensrepräsentationssprache Loom – einer Erweiterung von KL-ONE – basiert. Damit wird eine hierarchische terminologische Wissensbank spezifiziert, die das Domänenmodell darstellt. Mit diesem Modell werden wiederum die Inhalte der zu integrierenden Quellen unabhängig voneinander beschrieben, im Wesentlichen durch *is-a*-Beziehungen zwischen den globalen und den lokalen Konzepten. Auf diese Weise wird ein LaV-Ansatz realisiert.

Der in [GBMS99] beschriebene Context-Mediator verwendet ebenfalls ein Domänenmodell, das hier eine Menge von primitiven und semantischen Typen umfasst. Instanzen von semantischen Typen können dabei unterschiedliche Werte in verschiedenen Kontexten haben, d.h. in den einzelnen Quellen können verschiedene Annahmen über die Interpretation der Werte getroffen werden. Mit den Quellen lassen sich Kontextaxiome assoziieren, die die Konvertierung der lokalen Werte in das globale Domänenmodell spezifizieren.

Dem hier vorgestellten YACOB-System am nächsten kommt der XML-Mediator STYX [ABFS02], der ebenfalls auf dem LaV-Prinzip basiert und eine Ontologie als Integrationsmodell nutzt. Die Quellenbeschreibungen erfolgen durch XPath-Ausdrücke. Als Anfragesprache wird eine einfache OQL-ähnliche Sprache unterstützt, wobei die Konzepte Klassen entsprechen. Die einzige Operation auf Konzeptebene ist die Traversierung von Beziehungen.

Im Vergleich mit diesen Ansätzen kann das in diesem Beitrag vorgestellte YACOB-Mediatorsystem der semantischen Integration zugeordnet werden, wobei das LaV-Prinzip verfolgt wird. Im Gegensatz etwa zu SIMS erfordert der YACOB-Mediator jedoch keine explizite Modellierung der Quelleninhalte, sondern diese werden den Konzepten des Domänenmodells in Form von Abbildungsbeschreibungen zugeordnet. Die speziellen Kategorieklassen des YACOB-Konzeptmodells lassen sich mit den semantischen Typen des Context-Mediators vergleichen, allerdings werden die dort verwendeten Axiome in YACOB vereinfacht durch Wertabbildungen ersetzt. Die Verwendung semantischer Typen ist jedoch primär auf die Behandlung von Attributwertkonflikten ausgerichtet und nicht wie

beim hier beschriebenen Ansatz auf die Überwindung semantischer Heterogenität bei der Integration. Eine weitere Besonderheit des YACOB-Mediators ist die Unterstützung von Schemaoperationen durch die Anfragesprache CQuery. Dies erlaubt nicht nur die Ermittlung von Properties zur Laufzeit sondern auch die „Berechnung“ von Konzepten, zu denen Daten aus den Quellen angefragt werden sollen.

Gerade im Zusammenhang mit Mediatoren wurden in letzter Zeit Techniken zum Semantic Caching entwickelt. Dabei wir das Caching statt auf physischer Ebene, auf Datenebene zusammen mit Daten zur Beschreibung der Cache-Einträge durchgeführt [DFJ+96]. Diese Technik kommt insbesondere bei der Datenintegration [ASPS96] und im World Wide Web [LC01] zum Einsatz.

Hauptanliegen der Bemühungen zum Semantic Web ist die Verbesserung der automatisierten Verarbeitbarkeit von Web-Informationen durch Hinzufügen von den Inhalt beschreibenden Metainformationen – d.h. die Interoperabilität auf semantischer Ebene zu erreichen. Eine Schlüsselrolle spielen dabei Ontologien. Als einfachste Form kann in diesem Zusammenhang das hier verwendete RDFS angesehen werden. Für RDF und RDFS wurden auch bereits diverse Anfragesprachen vorgeschlagen, welche die Besonderheiten von RDF-Beschreibungen etwa im Vergleich zu einfachen XML-Dokumenten berücksichtigen [MKA+02]. Allerdings sind RDF-Anfragesprachen nur auf die Metadatenebene beschränkt und können so allenfalls – wie in Abschnitt 6 gezeigt – als eine Teilkomponente eines konzeptbasierten Anfragesystems angesehen werden. Weitergehende Ansätze für Ontologiesprachen wie DAML+OIL [Hor02] bieten Erweiterungen zu RDFS, wie Klassenkonstruktoren, Axiome, mit denen Subsumption oder Äquivalenz von Klassen und Eigenschaften ausgedrückt werden können, und Inferenzmechanismen. Andere Beispiele für die Nutzung von Ontologien zur Informationsintegration sind u.a. Portale [MSS+02] oder semantische Suchmaschinen wie OntoBroker [DEFS99].

9 Zusammenfassung und Ausblick

Die Repräsentation von Hintergrundwissen in Form von Konzepten und deren Beziehungen als semantische Metadaten von Mediatoren kann in Verbindung mit einem Local-as-View-Ansatz die Integration neuer Quellen sowie die Formulierung von globalen Anfragen vereinfachen. Vor diesem Hintergrund wurden in diesem Beitrag ein Anfragesystem zu einer konzeptbasierten XML-Anfragesprache vorgestellt und die Schritte der Transformation und Ausführung von Anfragen beschrieben. Als Integrationsmodell wurde dabei RDF Schema gewählt, wodurch die Nutzung verfügbarer Werkzeuge zur Modellierung und zum Datenaustausch möglich ist.

Das gewählte Anwendungsgebiet – die Integration kulturhistorischer Internet-Datenbanken – wird speziell durch eine Benutzerschnittstelle berücksichtigt, die eine inkrementelle Anfrageverfeinerung erlaubt und durch einen Anfrage-Cache unterstützt wird. Mit dem implementierten Prototypen ist der Zugriff auf die Internet-Datenbanken *www.lostart.de* über eine interne JDBC-Schnittstelle, die vom Web Service genutzt wird, sowie auf *www.herkomstgezocht.nl* über einen Wrapper möglich.

Literatur

[ABFS02] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-Based Integration of XML Web Resources. In *ISWC 2002*, LNCS 2342, pages 117–131. Springer-Verlag, 2002.

[ACHK93] Y. Arens, C.Y. Chee, C.-N. Hsu, and C.A. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[ASPS96] S. Adah, K. Selcuk Candan, Y. Papakonstantinou, and V.S. Subrahmanian. Query Caching and Optimization in Distributed Mediator Systems. In *SIGMOD'96*, pages 137–148, 1996.

[BGL+99] C.K. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-Based Information Mediation with MIX. In *SIGMOD'99*, pages 597–599, 1999.

[DEFS99] S. Decker, M. Erdmann, D. Fensel, and R. Studer. OntoBroker: Ontology-based Access to Distributed and Semi-Structured Information. In *DS-8: Semantic Issues in Multimedia Systems*. Kluwer, 1999.

[DFJ+96] S. Dar, M.J. Franklin, B.T. Jónsson, D. Srivastava, and M. Tan. Semantic Data Caching and Replacement. In *VLDB'96*, pages 330–341, 1996.

[GBMS99] C.H. Goh, S. Bressan, S.E. Madnick, and M.D. Siegel. Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. *ACM Transactions on Information Systems*, 17(3):270–293, 1999.

[GMPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J.D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[Hal01] A.Y. Halevy. Answering Queries using Views: A Survey. *The VLDB Journal*, 10(4):270–294, 2001.

[Hor02] I. Horrocks. DAML+OIL: a Description Logic for the Semantic Web. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pages 4–9, 2002.

[LC01] D. Lee and W.W. Chu. Towards Intelligent Semantic Caching for Web Sources. *Journal of Intelligent Information Systems*, 17(1):23–45, 2001.

[LGM01] B. Ludäscher, A. Gupta, and M.E. Martone. Model-based Mediation with Domain Maps. In *ICDE'01*, pages 82–90, 2001.

[LRO96] A.Y. Levy, A. Rajaraman, and J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB'96*, pages 251–262. Morgan Kaufmann, 1996.

[MKA+02] A. Magkanaraki, G. Karvounarakis, Ta Tuan Anh, V. Christophides, and D. Plexousakis. Ontology Storage and Querying. Technical Report 308, Foundation for Research and Technology Hellas, Institute of Computer Science, April 2002.

[MSS+02] A. Maedche, S. Staab, R. Studer, Y. Sure, and R. Volz. SEAL – Tying Up Information Integration and Web Site Management by Ontologies. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pages 10–17, 2002.

[RS97] M.T. Roth and P.M. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *VLDB'97*, pages 266–275, 1997.

Catalog Integration Made Easy*

Pedro José Marrón, Georg Lausen and Martin Weber
Universität Freiburg, Institut für Informatik
Georges-Koehler-Allee, Geb. 51
79110 Freiburg, Germany
{pjmarron, lausen, weber}@informatik.uni-freiburg.de

Abstract: In this paper, we study adaptive evaluation techniques for querying XML-based electronic catalogs, and show, by means of experiments performed on real-world catalogs, that our approach can be used to integrate them with virtually zero effort at start-up time, and a small constant factor needed to perform the adaptive evaluation for all subsequent queries. We reach the conclusion that, from a strictly technical perspective, the classic role of the global catalog can be assumed in an ad-hoc manner by any catalog that forms part of a collaborative federation of XML-based catalogs, and implements our adaptive query algorithms, independently of the storage model used to access its contents.

Keywords: E-Commerce, E-Catalog, Catalog integration, XML, XPath

1 Introduction

The proliferation of XML-based catalog research [FLM98, Jhi00, Kel97], has made it easier for companies and suppliers to integrate their offers into common catalogs that allows them to reach customers in an easier way. Despite improvements in that area, the problems found in the classical view approach to catalog integration [SH01] have not been solved. The fact that the integration of different catalogs implies the generation, maintenance and adaptation of a global catalog that forwards the appropriate queries to the corresponding local catalogs for evaluation, requires us to perform query rewriting in order to provide the user with an answer.

In this paper, we evaluate, extend and improve on the techniques developed in [LM02] for querying XML-based electronic catalogs, that eliminates the need to perform an explicit query rewriting. Our methodology uses an adaptive query evaluation strategy that allows us to perform the same query on all local catalogs and still obtain accurate answers.

The rest of this paper is organized as follows. Section 2 explains the gist of our algorithms, introducing the types of problems found, but not addressed, in our previous work. In section 3, we refine our approach in order to improve on the deficiencies of the basic

* A short version of this paper has been accepted as a poster contribution for ICDE 2003 to be held in March 2003 in Bangalore, India.

system, leaving for section 4 the experimental evaluation of both the basic and refined systems. Finally, section 5 deals with related work in this area, and section 6 concludes this paper.

2 Adaptive Query Evaluation

Our approach to querying XML-based catalogs adaptively relies on the following three pillars: (1) Since, according to the XPath query model, an XPath query is evaluated by decomposing it into smaller, independent pieces, each of them can be transformed on-the-fly and adapt to the current catalog structure; (2) the use of a flexible fitness function allows us to discriminate more accurate solutions from others; and (3) the hierarchical nature of the conceptual organization of product catalogs makes it possible to eliminate or introduce concept categories at certain levels without affecting the outcome of the query.

Let us first revise the classic evaluation model found in the XPath standard [CD99].

2.1 XPath Evaluation Model

A formal characterization of this model, originally proposed in [ML01], can be summarized as follows:

Definition (XPath Query) An XPath Query Q_X is defined as $Q_X = q_0/q_1/\ldots/q_n$, where q_i is an XPath subquery defined below, and '/', the XPath subquery separator. □

Definition (XPath Subquery) An XPath Subquery q_i is a 3-tuple $q_i = (C_i, w_i, C_{i+1})$, where: C_i is a set of XML nodes that determine the input context; w_i is the Path Expression to be applied to each node of the input context (defined below); and C_{i+1} is a set of XML nodes resulting from the application of the path expression w_i onto the input context C_i. C_{i+1} is also called the output context.

An XPath subquery is also called a *location step* in the terminology used by the World Wide Web Consortium [W3C]. □

Definition (XPath Path Expression) A Path Expression w_i is a 3-tuple $w_i = a_i :: e_i[c_i]$ such that: a_i is an axis along which the navigation of the path expression takes place[1]; e_i is a node expression that tests either the name of the node or its content type; and c_i is a boolean expression of conditional predicates that must be fulfilled by all nodes along the path. □

[1] Only the `child` and `parent` axis are considered in our adaptive evaluation algorithm

2.2 Adaptive Query Strategy

The adaptive query strategy involves two main steps: a preprocessing step, where, if needed, each individual concept found in the query is translated based on the elements present in the local catalog; and the actual processing of the query.

Each local catalog is expected to provide a mapping from the concepts found in the global catalog to its own local representation, so that discrepancies that might appear as a result of using synonyms, or other languages for the same concepts are easily solved. For the purposes of our algorithms, we assume that this mapping is provided by translating the set of concepts transmitted as part of the query without paying attention to either structural or semantical relationships between the original query and the local catalog contents.

After the preprocessing step has completed, our algorithm performs the following steps: (1) Application of each adaptive transformation on the input context of each subquery; (2) evaluation of the fitness function at each step to provide a node ranking used to discriminate one set of solutions over another.

Definition (Subquery Transformations) The three possible transformations performed on a subquery q_i by the first step of our algorithm are: *No transformation (n):* Where the query is evaluated as it was originally specified by the user; *subquery generalization (g):* Where the axis of a particular subquery is augmented according to the following rules: if the original axis (a_i) is `child`, the augmented axis (a_i') becomes `descendent`. Similarly, if a_i is `parent`, a_i' becomes `ancestor`; *subquery elimination (e):* A subquery is eliminated from the original XPath query if its result set (output context) is empty, allowing for the further evaluation of the following queries as if it had never been on the original query. □

In order to determine the correct sequence of transformations to apply at each subquery, given a subquery q_i with input context C_i, we evaluate it with respect to each subquery transformation, so that from each input context C_i, we generate three output contexts: C_{i+1}^n, C_{i+1}^g and C_{i+1}^e that correspond, respectively, to the application of the *no transformation (n), subquery generalization (g)* and *subquery elimination (e)* transformations defined above. Each context $C_{i+1} = C_{i+1}^n \cup C_{i+1}^g \cup C_{i+1}^e$ is formed by the union of the result contexts obtained by the application of each one of the three transformations.

In order to distinguish the nodes that compose the optimal solution to the query we now define a fitness function that assigns a metric on the nodes based on the type of transformation used to generate them. Let us now define the form of the (global) fitness function that was originally proposed in [LM02], and that will be used later in comparison to other variants for the experiments of section 4.

Definition ((Global) Fitness function) Let q_i be the current subquery to be evaluated, and C_i its associated input context. Each node $n \in C_i$ is augmented to have a value v_n resulting from the evaluation of the previous subqueries $q_0 \ldots q_{i-1}$.

Then, the fitness function assigns a value v_m to each new node in $m \in C_{i+1}$ generated by a node $n \in C_i$ as follows: if $m \in C_{i+1}^n$, then $v_m = b^2 + v_n$; if $m \in C_{i+1}^g$, then $v_m = b + v_n$; and if $m \in C_{i+1}^e$, then $v_m = 1 + v_n$. b is a positive integer that represents the base of the fitness function.

If a node n has been generated by the application of more than one strategy, the final context C_{i+1} only contains the instance of n with the biggest value v_n; finally $v_{root} = 1$.□

As can be deduced from the definition of the fitness function, nodes generated as a result of the evaluation of a subquery in its original state have a higher fitness value than those generated by the generalization of the subquery, or those obtained by its elimination. The reason behind this choice is to favor the original query over transformation operations that might abstract away the intention of the user.

2.3 Formal Definition and Analysis

The distinction between the three types of transformations implies that the fitness function we just described provides an efficient way to traverse the partial order formed by the evaluation process of a particular query.

In fact, since at each step of the query evaluation we keep the results generated by each transformation, the contents of the last context at the end of the computation (as defined in subsection 2.2), correspond to the set of nodes found performing a series of *no transformations* ($nn \ldots n$), another set that corresponds to always performing *elimination* ($ee \ldots e$), and many other possibilities in between.

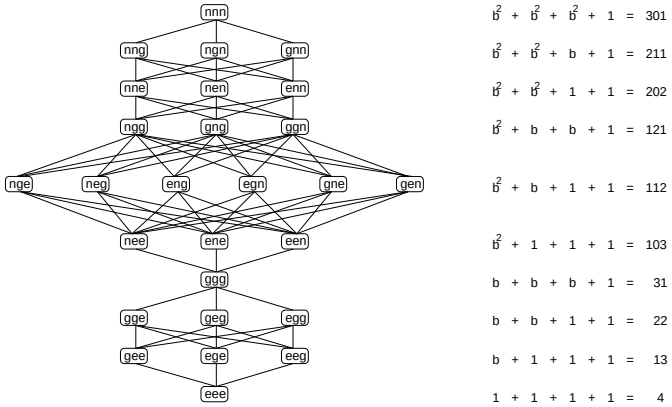


Figure 1: Partial Order for $q_l = 3$ and $t = 3$

Figure 1 shows the partial order generated for a query of length three ($q_l = 3$), by the application of the global fitness function defined in the previous section for $b = 10$, assuming that the number of allowed transformations is three ($t = 3$). As it is shown on the right-hand side of Figure 1, each level is assigned a different value by the fitness function, so that levels closer to nnn have higher values than other levels further down in the ordering.

Of course, it is not clear whether or not it is correct to say that $nne > ngg$, or that $nee > ggg$, as determined by our fitness function. In any case, given such a partial order, we can reorganize each level at our convenience and extract another fitness function that maps the new structure, as long as the new partial order does not contain any contradictions in the inequalities it defines. For example, assuming that the fitness function simply adds

the fitness values of each transformation, the ordering of Figure 1 determines that $3 \cdot n > 2 \cdot n + g$, $2 \cdot n + g > 2 \cdot n + e$, and so on, which leaves us with the following inequalities: $n > g$, $g > e$, $n - 2 \cdot g + e > 0$ and $n - 3 \cdot g + 2 \cdot e > 0$.

For higher level orderings, where the length of our query is q_l, keeping the ordering of nne and ngg as in the figure, the partial orderings must satisfy the following series of inequalities: $n > g$, $g > e$, $n > e$, $n > 2 \cdot g - e$, $\ldots n > q_l \cdot g - (q_l - 1)e$. Therefore, solving for n, g and e for the general case, and assuming that e takes on a constant value c ($e = c$), we have $g = q_l + c$ and $n = q_l^2 + c$. Observe that this happens to correspond to the original definition of our fitness function, assuming that $e = 1$ and $q_l < 10$, that is, the length of queries is smaller than 10.

3 Algorithmic Refinements

There are several limitations to the original approach, that have lead to the refinement and consequent improvement of our methodology. They can be classified in the following categories: *Fitness function limitations*, *semantic limitations* and *structural limitations*. Let us now describe in detail the changes performed to our algorithms to try to improve on this deficiencies.

3.1 Fitness Function Improvements

The definition of the fitness function given in the previous section has the limitation, that its base b is global to the catalog and therefore, each node is weighted equally independently of its location within the product category tree. This contradicts the usual way product catalogs are structured, because normally, elements found at lower levels in the tree denote more specific products than those found at higher levels. This specificity factor is not taken into account unless we redefine our fitness function to incorporate a base $b_l > 0$ that assigns a greater value to nodes found at lower levels:

Definition ((Level) Fitness Function) Let q_i be the current subquery to be evaluated, and C_i its associated input context. Each node $n \in C_i$ is augmented to have a value v_n resulting from the evaluation of the previous subqueries $q_0 \ldots q_{i-1}$. Then, the fitness function assigns a value v_m to each new node in $m \in C_{i+1}$ generated by a node $n \in C_i$, such that: if $m \in C_{i+1}^n$, then $v_m = b_l^2 + v_n$; if $m \in C_{i+1}^g$, then $v_m = b_l + v_n$; and if $m \in C_{i+1}^e$, then $v_m = 1 + v_n$. $b_l > 0$ is a level-dependent positive integer that represents the value of the fitness function base at each level in the product catalog. □

This small change has the particular effect of increasing the number of levels the query partial order has since, for example in the ordering of Figure 1, nge and gne would be assigned to different levels. Following the same reasoning, however, we could argue that some catalogs contain nodes at the same level that should be weighted differently. Therefore, a more involved representation of the fitness function, but also more flexible, would allow b, the base of the fitness function, to be dependent on the specific node it is applied

upon:

Definition ((Node) Fitness Function) b, the base of the fitness function, is now dependent on the node. Therefore, $b = f(m)$, where $f(m) > 0$ is a node-specific function that returns a positive integer that represents the value of the fitness function base for the node m in the catalog. □

Again, this definition has the effect of thinning out the query partial order, providing us with extra information that allows to distinguish between the different permutations at a particular level in the ordering. The implementation of such a function is not so straightforward, since we need a way to determine the weight of each node. The two strategies that come to mind can be classified as follows:

Hand-crafted functions: Where the maintainer of the local catalog decides based on experience what the weight for each node should be. *Learning functions:* Where the function modifies itself based on the kind of queries performed on the local catalog.

Finally, the fitness function could take into consideration the structure of the global catalog. Since the purpose of our algorithm is to avoid any kind of communication between the global and the local catalogs except for the specific query that needs to be evaluated, the only way for a local catalog to incorporate any kind of information from the global catalog is through analysis of the query received to be evaluated. Therefore, the final version of the fitness function we consider in this paper, is a query-specific fitness function that assigns a higher weight to subqueries further down in the query specification.

Definition ((Query) Fitness Function) b, the base of the fitness function, depends on the position of a specific subquery. Therefore, $b = f(q_i)$, where $f(q_i) > 0$ is a query-specific function that returns a positive integer representing the value of the fitness function base for the i^{th} subquery, such that: $f(q_i) < f(q_j) \iff 0 <= i < j <= n$ □

Therefore, this definition of the fitness function has the effect of transforming our query partial order into a *total linear order*, where each pair of query evaluations can be compared to each other.

3.2 Semantic Extensions

There is a class of "problematic nodes" in product catalogs (from this point on called *semantically weak nodes*), that, under certain conditions, lead to non-optimal results and that, although resolved by the application of a correct node-specific fitness function, can be explained best by referring to semantic information.

Semantically weak nodes are nodes that by themselves do not convey enough information to be a distinguishing product, since there are other nodes in the system with the same name. For example, it does not make any sense to talk about `accessories` without knowing whether or not we are talking about `CPU accessories`, or `printer accessories`, etc. By assigning a smaller value for the fitness function base evaluated on such nodes, we can improve on the results obtained at such nodes, without negatively affecting the outcome of other queries, as it has been verified by the experiments detailed in section 4.

3.3 Improved Structural Adaptability

Since the original implementation of our algorithm only takes into consideration element nodes, and not attributes, we need to incorporate a new type of transformation that allows us to also include attributes in our catalog search: the subquery *upgrade* transformation. An attribute node needs to be upgraded, that is, replaced by its corresponding parent element, if there is a match between either the name or the value of such attribute and the specification of a particular subquery.

The application of this operation allows us to correctly evaluate the XPath query $Q_{global} = /a/b/c$ by effectively translating it to $/a[d = 'b']/c$ or $a[b = '*']/c$ as needed.

4 Experimental Evaluation

In order to prove the feasibility of our approach, we have performed experiments on top of reduced versions of real XML catalogs provided by several product resellers on the Internet.

All experiments have been performed on a Pentium class computer running Linux and the OpenLDAP [2] server. This query processing system, described in [ML01], is able to process XPath queries on top of arbitrary XML documents stored in an LDAP [WHK97] database, but our results are independent of the actual storage mechanism used and are, therefore, only a function of the adaptive algorithms explained throughout this paper.

In order to test our algorithms, we have visited the homepages of Reichelt (*Rei.*), Alternate (*Alt.*) and K & M Elektronik (*K & M*) [3], three real-world companies that sell electronics and computer products online, and constructed XML catalogs from the data found in their site. The fourth catalog used in the experiments is an artificially crafted catalog (*Art.*), that heavily uses attributes to encode the type of information that is found in the other three catalogs under different elements and product categories. Finally, we have combined their DTDs into a virtual global catalog that contains no data, by inspecting the products offered by each company, and providing a product hierarchy that encompasses all products. Table 1 contains some structural data for each catalog. The column "Nodes" represents the total number of nodes in the catalog, "Depth" the maximum length from the root to the leaves, and "Outdeg", the maximum number of children a particular node has.

Catalog	Nodes	Depth	Outdeg
Global	129	4	16
Alt.	299	5	18
Rei.	89	5	9
K & M	136	4	22
Art.	27	3	6

Table 1: Catalog Sizes and Structures

Type of Fitness Function	Semantic	
	No	Yes
Global, $b = 10$	GN	GS
Level-specific, $1 \leq b \leq 10$	LN	LS
Node-specific (learning), $1 \leq b \leq 10$ (obtained by averaging the frequency of queries)	NN	NS
Query-specific, $1 \leq b \leq 10$ (determined by the position of a specific subquery)	QN	QS

Table 2: Types of Fitness Functions

[2] http://www.openldap.org

[3] http://www.reichelt.de, http://www.alternate.de, http://www.kmelektronik.de

4.1 Adaptive Algorithm Testing

The purpose of this set of experiments is to determine the level of correctness of our algorithms, given that they use the types of fitness functions and semantic extensions detailed in the previous sections.

	No semantic					Semantic				
	Alt.	Rei.	K & M	Art.	**Overall**	Alt.	Reich.	K & M	Art.	**Overall**
Global:	2.8%	0.3%	4.8%	1.8%	**2.4%**	0.6%	0.3%	2.6%	0.0%	**0.9%**
Level:	2.7%	0.3%	2.6%	1.8%	**1.8%**	0.5%	0.3%	2.6%	0.0%	**0.9%**
Query:	2.4%	0.1%	3.1%	1.8%	**1.8%**	0.1%	0.3%	0.5%	0.9%	**0.5%**
Node:	2.3%	0.1%	3.4%	1.0%	**1.7%**	0.4%	0.3%	1.3%	0.0%	**0.4%**

Table 3: Error Rates

Table 2 shows the types of fitness-functions which we applied in our experiments. We evaluated all of these fitness-functions with the semantic extension (*semantic=yes*) described in section 3.2; and without it (*semantic=no*). The functions are able to perform the transformations described in [LM02], plus the *upgrade* operation described in section 3.3.

Table 3 shows the error rate for each type of function detailed by catalog and the overall error rates found by performing 783 queries on top of the four local test catalogs. In it we can see that the error rates vary between 3.5% and 0.4%, with accuracy going up to 100% for all catalogs if we consider the second highest rated nodes also part of the solution.

In a second set of experiments, we submitted to three of the four local catalogs every possible query that could be generated by the remaining catalog. As it can be seen in table

	GS Function				QS Function			
	Alt.	Rei.	K & M	Art.	Alt.	Rei.	K & M	Art.
Alt:	–	1.1%	2.3%	0.9%	–	0.3%	0.7%	0.9%
Reich:	4.8%	–	2.4%	0.8%	1.4%	–	0.4%	0.8%
K&M:	1.6%	0.0%	–	0.0%	0.2%	0.0%	–	0.7%
Art:	0.0%	0.0%	6.0%	–	0.0%	0.0%	0.0%	–

Table 4: Cross-Evaluation Catalog Error Rate

4, the error rate for two of the tested functions (*GS* and *QS*), indicates that our algorithms present exactly the same behavior and incur in approximately similar error rates.

4.2 Error classification

The types of queries performed on the system that do not lead to an optimal result are of two types: (1) Queries performed on *semantically weak nodes* where the catalog lacks specific products that exactly match those of the weak node; and (2) queries whose correct result is debatable, that is, not even a theoretically perfect rewriting algorithm would be able to automatically determine whether or not the solution proposed by our algorithm is correct without the catalog maintainer making a somewhat informed decision.

The first type of errors can be corrected by incorporating some type of semantic information to direct the search, whereas the second type of errors must be corrected by the appropriate tweaking of the fitness function for those queries/nodes that do not produce

the desired result. The flexibility of the fitness function allows for such modifications without the need to change the mechanism of our algorithms.

4.3 Result Analysis

From the two result tables, we can conclude that our algorithm is very insensitive to structural differences. Even though the topology of the *Alternate* catalog is much closer to that of the global catalog than the *Reichelt* catalog, the error rate for *Reichelt* is significantly smaller. The main reason is the absence of *semantically weak nodes* in the *Reichelt* catalog, leading us to conclude that the accuracy of our algorithm is dependent on the clean and clear specification of product names and categories.

Furthermore, the results obtained from the second set of experiments performed on our catalogs, indicate that from a purely technical perspective, there is no need to have a global catalog to centralize our queries, since our algorithms are able to deal with the most varied structural differences without a degradation in performance, as measured by the error rate.

5 Related Work

The integration of catalogs is discussed, from a very broad perspective in [SH01], where the authors point out some of the challenges integration software is usually faced with. The authors indicate that integration software should be based on XML and be able to process XPath queries now, and possibly XQuery in the future.

Similar goals for the easy evaluation of queries on top of XML documents are described in [SKW01], where the authors describe the idea of nearest concept queries to allow for query processing on documents whose mark-up structure is not known. The difference lies on the specifics of their methodology, since they involve the use of a *meet* operator and the evaluation of regular path expressions, whereas our algorithms exploit query transformations to achieve their goal.

Interestingly enough, the need for additional query formulation techniques have lead other researches towards also using regular path expressions [AQM+97, FS98], but in our opinion, a graphical interface for user navigation within the tree structure would just suffice for these purposes, given the adaptive evaluation techniques described in our work, and that typical catalog users cannot be expected to write regular path expressions on their own.

6 Conclusion

In this paper we have extended, improved and evaluated a set of adaptive XPath evaluation techniques by including the processing of semantically weak nodes, as well as the definition of a new type of transformation that allows us to transform attributes into elements.

We have also provided a formal characterization of our fitness function definitions, as well as experimental results that show the applicability of our adaptive algorithm on real-world data, while at the same time, categorizing the types of errors our system could produce if certain unfavorable conditions are met, so that catalog maintainers can avoid them. Furthermore, we have reached the conclusion that, from a strictly technical point of view, the role of the global catalog could be assumed by any local catalog.

To the best of our knowledge, there is no other XPath evaluation system used for XML-based catalog integration that is able to adapt to the specific structure of the catalogs in use without requiring some form of a rewriting algorithm to transform a query performed on a structurally different catalog, while still providing the level of accuracy our system does.

References

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digial Libraries*, 1(1):68–88, 1997.

[CD99] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. `http://www.w3c.org/TR/xpath`, November 1999.

[FLM98] Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database Techniques for the World-Wide-Web: A Survey. *Sigmod Record*, 27(3), September 1998.

[FS98] Mary F. Fernández and Dan Suciu. Optimizing Regular Path Expressions Using Graph Schemas. In *Proc. of the 14th Intl. Conf. on Data Engineering*, Orlando, FL., USA, Feb. 1998.

[Jhi00] A. Jhingran. Moving up the food chain: Supporting E-Commerce Applications on Databases. *Sigmod Record*, 29(4), December 2000.

[Kel97] A.M. Keller. *Readings in Electronic Commerce*, chapter Smart Catalogs and Virtual Catalogs. Addison Wesley, 1997.

[LM02] Georg Lausen and Pedro José Marrón. Adaptive Evaluation Techniques for Querying XML-based E-Catalogs. In *Proc. of the 12th Intl. Workshop on Research Issues on Data Engineering*, February 2002.

[ML01] Pedro José Marrón and Georg Lausen. On Processing XML in LDAP. In *Proc. of the 27th Intl. Conf. on Very Large Data Bases (VLDB)*, Rome, Italy, September 2001.

[SH01] Michael Stonebraker and Joseph M. Hellerstein. Content Integration for E-Commerce. In Walid G. Aref, editor, *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data*, Santa Barbara, California, May 2001.

[SKW01] Albrecht Schmidt, Martin Kersten, and Menzo Windhouwer. Querying XML Documents Made Easy: Nearest Concept Queries. In *Proc of the 17th Intl. Conf. on Data Engineering*, Heidelberg, Germany, April 2001. IEEE Computer Society.

[W3C] W3C – The World Wide Web Consortium. `http://www.w3c.org/`.

[WHK97] M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3). RFC 2251, Dec, 1997.

XPath-Aware Chunking of XML-Documents

Wolfgang Lehner
(EMail: wolfgang@lehner.net)

Dresden University of Technology
(Database Technology Group)
Dürerstr. 26, D-01062 Dresden

Florian Irmert
(florian@irmert.de)

University of Erlangen-Nuremberg
(Database Systems)
Martensstr. 3, D-91058 Erlangen

Abstract

Dissemination systems are used to route information received from many publishers individually to multiple subscribers. The core of a dissemination system consists of an efficient filtering engine deciding what part of an incoming message goes to which recipient. Within this paper we are proposing a chunking framework of XML documents to speed up the filtering process for a set of registered subscriptions based on XPath expressions. The problem which will be leveraged by the proposed chunking scheme is based on the observation that the execution time of XPath expressions increases with the size of the underlying XML document. The proposed chunking strategy is based on the idea of sharing XPath prefixes among the query set additionally extended by individually selected nodes to be able to handle XPath-filter expressions. Extensive tests showed substantial performance gains.

1 Motivation

XML has gained the status of a de-facto standard for wrapping (semi-) structured data and exchanging it via the Internet. Even web surfing, i.e. requesting an HTML page from a web server, implies the point-to-point transfer of an XML document as the payload of an HTTP response, if the XML document follows the standardized XHTML schema definition. Reversing the communication pattern of this simple request/response yields the publish/subscribe pattern to build large scale information dissemination systems ([BeCr92], [FoDu92], [AAB+98], [FJL+01], [BaWi01]). In this scenario, data producers (publishers) on the one hand are exposing data to an information broker. On the other hand, users interested in receiving notifications regarding information about specific topics from potentially anonymous data producers are placing a subscription at the broker. As soon as a data fragment enters the brokering component, all registered subscriptions are evaluated against the incoming data. In the end, only those subscribers with a matching subscription are notified by routing the interesting part of the original message to the corresponding subscriber. Obviously the matching component, comparing registered subscriptions to an incoming data fragment reflects the core of an efficient publish/subscribe system. Based on XML documents as messages being exchanging between publishers and subscribers and XPath expressions as a mean to specify subscriptions, we propose a chunking framework to speed up the filtering process and cut down the time needed for matching subscriptions against incoming information.

Building an Efficient Information Dissemination Framework

The database oriented approach to set up an information dissemination system based on the publish/subscribe communication pattern may exploit the ECA model of a database triggering mechanism ([AgCL91], [HCH+99]): on inserting new (and very well structured) information into a database (the event), deliver the information (the action) if the subscription is satisfied (the condition). Unfortunately, the triggering model is primarily designed to perform complex actions (like checking integrity constraints) for a low quantity of triggers. Registering thousands of triggers (one for each subscription) to implement a large scale dissemination system does not sound feasible. Many extensions on a relational and/or semi-structured data model level, for example the concept of a ›Continual Query‹ in OpenCQ ([LiPT99])/NiagaraCQ ([CDTW00]) were made to reduce the pain of triggers. An alternative solution might be to see subscriptions as materialized views inside a database so that subscription evaluation is be reduced to the incremental maintenance of the corresponding subscription views. Exploiting the techniques of materialized views provides a transparent refresh of individual subscriber data and may additionally yield in internal optimizations like multiple query optimization ([LPCZ01]).

The other extreme of evaluating subscriptions is based on checking regular expressions on completely unstructured or semi-structured data (ASCII text, emails,...). Besides many systems in the information retrieval area, SIFT (Stanford Information Filtering Tool; [YaGa95]) was one of the first prominent filtering system published in the database community. SIFT was accompanied by Yeast ([KrRo95]), Siena ([RoWo97]), Gryphon ([ASS+99]), or the DBIS toolkit ([AAB+98]) focussing on different perspectives like routing messages in a distributed brokering environment or defining transformations of messages, etc.

The arrival of XML promised to close the gap between database oriented dissemination systems referring to well-structured information and pure filtering tools operating on any kind of data in evaluating regular expressions. Examples for XML-based dissemination systems can be found in [PFJ+01] (WebFilter), [AlFr00] (XFilter), and many more. Our filtering approach may be implemented on top of an XML-based dissemination system, if the following architectural requirements according to the publish/subscribe paradigm are satisfied (see figure 1):

- *publisher side*
  Before a publisher may expose information as XML documents to the information broker, the publisher has to register by submitting a schema definition. All following documents have to conform to this schema which is held at the broker.
- *subscriber side*
  After inquiring about the registered XML schemas, a subscriber may submit a subscription consisting of a complex XPath expression. XPath expressions of all subscribers are also stored locally at the broker. An incoming data stream is matched with the XPath expressions and a notification consisting of the interesting part of the original XML message is sent to the subscriber.

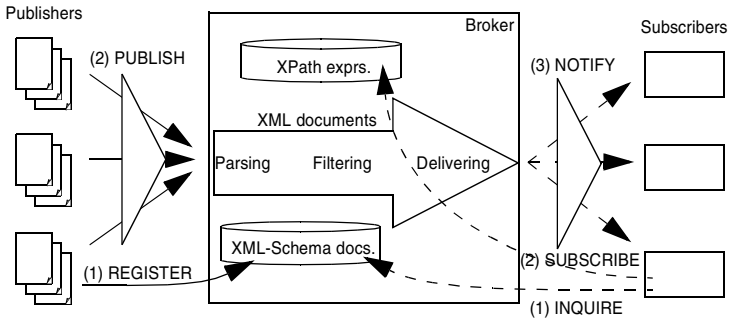


Fig. 1: Subscription Framework

The main idea of the approach discussed in this paper consists in using the schema information of the publishers to optimize the registered XPath expressions and produce a chunking scheme for incoming XML documents so that the filtering process at runtime may operate on multiple chunks instead of one single XML file. In a first step, the following section discusses existing filtering mechanism of XPath expressions, followed by the optimization steps performed during the prepare phase in section 3. Section 4 finally discusses the proposed chunking strategy. Section 5 provides the descriptions of results gained from performing extensive tests based on our ›eXtract‹ system implementation.

2 Related Work

XPath expressions ([BBC+01]) reflect a core building block in querying XML documents, either standalone or within an XQuery expression ([CFR+01]). An XPath expressions consists of two main features. A location step directly reflects a predicate within the hierarchical structure of an XML document where textual information is recursively wrapped by tags optionally holding additional attributes. Starting with a context node (either the root node of an XML document or any other well-specified node), an XPath expression identifies a set of subtrees of the original XML document. Each location step corresponds to a navigation step based on an axis within the document and the application of a predicate. Figure 2 illustrates the semantics of different location steps for a given context node and an hierarchical XML document tree.

The most prominent types of location steps are a ›single downward-step‹ (/ or /child::) and ›any number of downward-steps‹ (// or /descendant-or-self::node()/). For example /a/b retrieves all subtrees starting with a <b>-tag at the second level in the document directly under an <a>-tag. In the opposite, the expression /a//b results in all <b>-rooted subtrees somewhere in a part of the document starting with an <a> tag. Analogously to file systems, two dots (..) navigate to the next higher level.

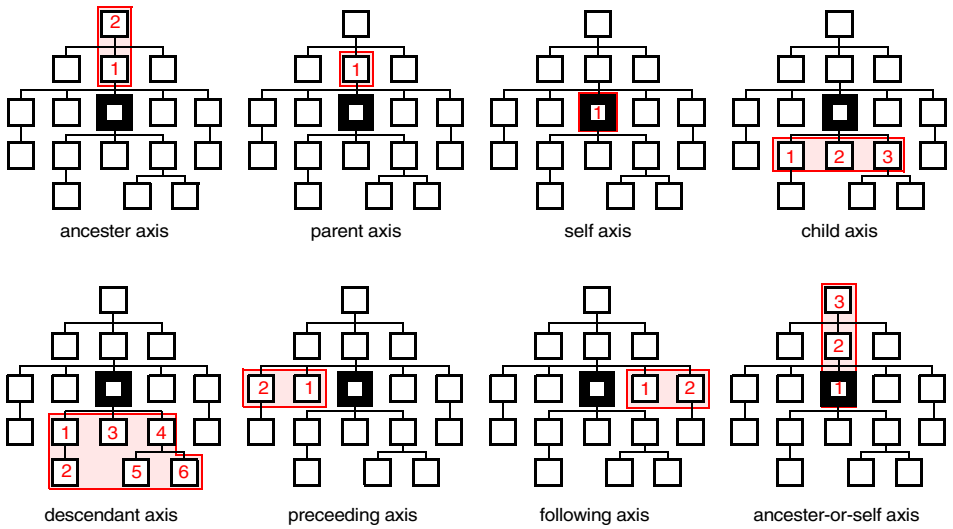


Fig. 2: XPath Location Steps

The application of predicates at a certain step may consist of a simple tag name (like in the example above), of a wildcard (*), and a filter specification within [ ]-parentheses. For example, /a/*/b returns a list of subtrees starting with <b> found at the third level originally rooted by an <a> tag. A filter expression may exhibit again multiple location steps and the reference of an attribute. For example the expression /a[b/c]/d returns all 2nd-level <d>-tag rooted subtrees if the corresponding parent <a>-tag holds a <c> tag as a grandchild with <b> as the direct descendant. Attributes are referenced by prefixing the attribute name with @. The above filter expression may be extended to hold an attribute z with value 'XML', yielding /a[b/c]/d[@z='XML'].

The XPath framework allows to address fragments of an XML document in a very complex manner and is either used isolated or as a core component in the XQuery language to specify more complex operations like joins and aggregates. To efficiently evaluate XPath expressions, multiple implementations and optimization are proposed in the literature. The classical way to evaluate XPath expressions is to create a main memory document object model (DOM; [WHA+00]) for a specific XML document and traverse the internal object structure (like Jaxen; http://www.jaxen.org). However many optimizations exist to speed up the XPath evaluation process.

On the one side, either special index structures like DataGuide ([GoWi97]), the approach of [LiMo01] or the APEX index ([ChMS02]) are proposed or existing multidimensional indexing technology like R-trees is used to support the efficient evaluation of XPath expressions ([Grus02]). On the other hand, filtering techniques are brought into discussion to index the queries, i.e. path expressions, instead of data ([MiSu99], [LaPa02]). The approach of [AlFr00] for example relies on the concept of a finite state machine so that processing a location step means switching a single transition in the machine. Each state con-

sists of a quadruple holding information regarding the current state of processing a single query. Query indexing considers / and // expressions including wild cards and the application of filters in the context of nested path expressions. Extending this idea, the approach of [CFGR02] performs substring decomposition on a syntactical level and defines common subpath expressions as clusters to reduce the number of states. Multiple clustered XPath expressions are additionally indexed using a specialized index structure.

The main idea of our contribution to speed up XPath filter expressions is to provide a preprocessing framework *in addition* to existing XPath filtering techniques by applying the following optimization steps:

- clean and transform registered XPath expressions during a prepare phase using the registered XML document schema.
- analyze XPath expressions and generate a chunking scheme for the incoming XML documents so that individual XPath expressions are evaluated on much smaller XML document fragments.
- exploit parallelism when executing XPath queries on XML data fragments without loosing any filtering capability.

The following section outlines the necessary transformations of the registered XPath expressions during prepare and runtime, while section 4 discusses the proposed chunking scheme.

3 The Basics: XPath Transformation

Once a subscriber registers an XPath expression at the information broker, the expression is added to the subscription database (Figure 1). In a single XPath filtering engine scenario, an incoming document is matched against the XPath expressions and the result is written into an XML result set document (Figure 3a). In the proposed way of applying XPath filters based on chunks, all XPath expressions are partitioned into smaller XPath sets using a stream-oriented interface; each set is then evaluated on a smaller XML document fragment which is sufficient to answer the allotted XPath expressions (Figure 3b).

Without any assumption of the underlying XPath evaluation method, it should be clear that applying less queries on a smaller XML document should speed up the overall filtering process. Even a sequential execution benefits from the preprocessing step done during the prepare phase. It is worth mentioning here that the *prepare phase* transforming XPath expressions and defining the chunking scheme for the XML documents does not influence the time needed to apply XPath filter expressions in a specific document. The time-critical *filtering phase* encompasses the creation of chunks and the evaluation of the registered XPath expressions. Moreover it should be noted that the chunking scheme may be incrementally adapted after a new subscription is added to the subscription base. The revised chunking scheme then applies to the next incoming XML document.

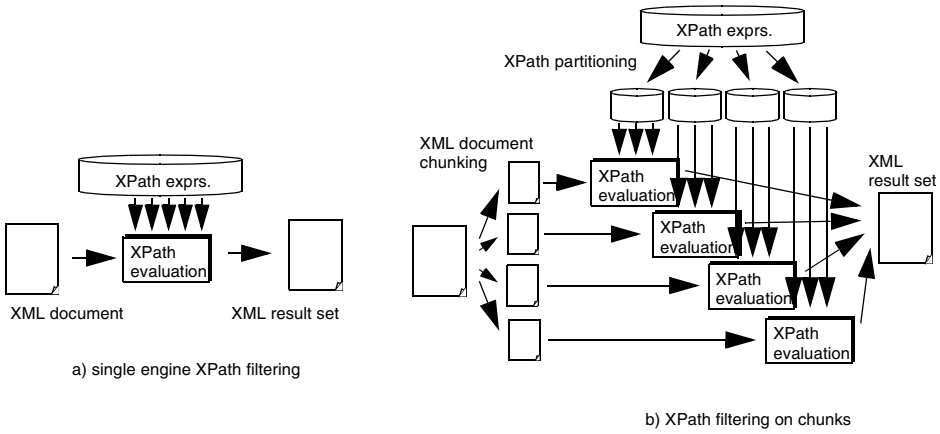


Fig. 3: Single Engine versus Chunked XPath Filtering

Cleaning and Transforming XPath Expressions

The preparing step of a set of XPath expressions applies transformation rules to convert all expressions into a standardized form, which is then subject of computing the chunking scheme. The overall goal is to generate XPath expressions with simple /-location steps and filters to provide a smooth foundation for the prefix-oriented chunking scheme.

In a first phase, all parent statements are eliminated by converting them to filter expressions. For example /a/b/../c identifies the <c>-nodes which comes after an <a>-node, if the <c>-node exhibits a <b>-node as his sibling. This may be equivalently written as /a[b]/c which may be read as: 'If an <a>-node has a <b>-node as a direct child, then give me the <c>-node children from the <a>-node.'

While the cleaning step is a pure syntactical transformation, the following step of resolving wildcards and //-expressions heavily relies on the existence of a schema definition. Therefore consider the sample DTD* given in figure 4 together with the following two expressions /a/b/*/c and /a/b/*/d. Without any further transformation the expressions would lead to the same prefix /a/b yielding no chunking criterion. However, referring to the XML schema, the resolution of the wildcards results in /a/b/l/c, a/b/m/c and a/b/k/d with potentially three different prefixes. Therefore, the overall goal in transforming XPath queries is to gain expressions with long prefixes so that the chunking scheme has many alternatives when deciding for a chunking point.

```
<!ELEMENT a (b+)  >
<!ELEMENT b (l*,m*,k*)>
<!ELEMENT l (c*)>
<!ELEMENT m (c*,a*)>
<!ELEMENT k (d*)>
<!ELEMENT d EMPTY>
<!ELEMENT c EMPTY>
```

Fig. 4: Sample DTD

* We usually rely on XML-schema but prefer the DTD version due to better readability and more compact presentation.

Considering the prefix problem in more detail, it is more important to resolve //-expressions, because expressions like //a or //b do not exhibit (at first sight) a common prefix. A answer to this problem is to find the first appearance of the nodes after an //-location step in each branch. For example, if – according to a given XML schema definition – a <d>-node appears within the document only in the constellation of /a/b/k/d, then we may transform //d into /a/b/k//d. Unfortunately, an XML document may exhibit an infinite depth if the corresponding schema exhibits a recursion. For example, the expression //c may be unrolled to /a/b/m/c or /a/b/m/a/b/m/c and so on. In this case, the transformation step expands the corresponding XPath expression until a recursion appears in the XML schema.

The last check performed during the cleaning and transforming step by matching each XPath expression against the corresponding XML schema is to eliminate expressions obviously evaluating to an empty result set.

In summary, consider the three sample expressions //c, /a/b/*/d and /a/d with regard to the sample DTD from figure 4. After transformation, the XPath set comprises the expressions /a/b/l//c and /a/b/m//c (resolving the //-expression) and /a/b/k/d (substituting wild cards). It is worth mentioning that the third sample expression (/a/d) is removed from the XPath set because the comparison with the corresponding schema does not provide any positive match. Furthermore, it should be noted that the resolution of a wildcard usually results in multiple possible XPath expressions.

4 The Chunking-Scheme

The overall goal to speed up the XPath filtering process by applying filter expressions on multiple smaller fragments of the original XML document requires an adequate partitioning scheme of the XPath expressions additionally implying a chunking scheme of the XML document. This partitioning mechanism depends on the number of chunks to be generated for the filtering process and the set of XPath expressions complying to the following rules:

- *even distribution*
  Each set of XPath expressions operating on the same chunk should have the same cardinality, i.e. all filtering expressions are supposed to be evenly distributed among the XML fragments.
- *small potpourri set*
  Since there may exist XPath expressions for which an assignment to a single chunk is not possible due to prefix incompatibility between chunk and XPath expression, we additionally keep a ›potpourri set‹ of XPath expressions. This set should be as small as possible, because usually less reduction is possible for the underlying XML document. For example if /a/b and /a/c determines the content of two separate chunks, an XPath expression /a/d would be assigned to the potpourri set. However,

since we explicitly consider filters in the chunking scheme (section 4), the expression /a/b[d]/e would be assigned to the chunk defined by /a/b, which in turn would be extended by the single nodes (not subtrees) addressed with /a/d.

Determine the Chunking Point based on XPath Query Trees

The algorithm to produce a chunking scheme for XML documents operates on a query tree, where each element of an XPath query is represented by a single node. The weight of a query tree node is initially set to 1 and increased with each additional XPath expression represented within the tree. For the sake of illustration, figure 5 shows the corresponding query tree after representing filter expressions /a/b/c, /a/b/d, and /a/b/c.

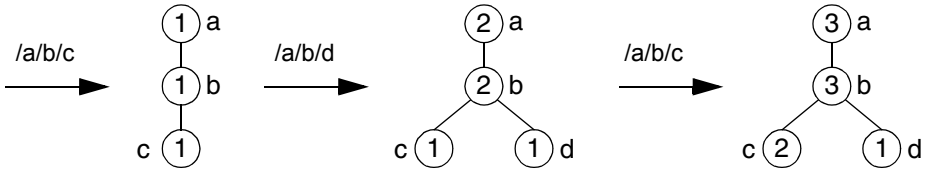


Fig. 5: Populating the Query Tree and Assigning Weigths

The naive approach to generate a chunking scheme would be to sort the nodes by their weight and take the TOP(n)-weighting nodes (with n as the number of chunks) as chunking criteria. In the example above, this would result in /a and /a/b. Unfortunately, since /a represents the root node, the first chunk would be the document itself. Moreover, /a/b also represents (more or less) the whole document, so that the second chunk would not result in any size reduction compared to the original XML document. Obviously the optimal solution for this scenario however would be chunks defined by /a/b/c and /a/b/d.

The revised chunking strategy is illustrated algorithmically in figure 6. To prevent the root node from being selected as a chunking prefix, it is removed from the set of possible chunking candidate nodes in a very first step. The second step consists in finding the node with the highest weight (uheavy) and the largest depth. This is done top-down by walking down a branch if a child has the same maximum weight so that we may get long prefixes finally resulting in small XML document fragments. This heaviest node is selected as a chunking candidate and added to the result list. To prevent the algorithm from picking the same prefix (or part of the prefix) in subsequent runs, we subtract the weight of the candidate node from all nodes (including the node itself) up to the root as long as the overall weight does not yield a negative value.

Referring to the sample query tree of figure 5, node /a/b would be selected as candidate node and added to the result list. The resulting scenario after reducing the weight of all nodes starting at /a/b up to the root (in this case /a) is depicted in figure 7a.

The next iteration in producing XML document chunks picks /a/b/c as a candidate node, because /a/b is no longer a valid choice and /a/b/c is the node with the highest weight. The aligning process of the weights within the query tree yields a reduction of node /a/b by the weight of /a/b/c (figure 7b). However, /a is not aligned, because its weight was already re-

```
Algorithm: GenerateChunkingScheme
Input:      D          // query tree node(name,weight)
            C          // number of chunks to generate
Output:     S          // nodes in the prefix tree representing chunks

BEGIN
    V=∅          // visited nodes
    S=∅          // result set of prefix tree nodes
    DO
        N = traverse(D) - {Root(B)}     // all nodes of the query tree without the root
        u_heavy = 0

        // search for next heaviest node not yet in the result
        FOREACH u ∈ (N-S)
            IF (weight(u) > weight(u_heavy))
                u_heavy := u
            END IF
        END FOREACH

        // while a child has the same weight, take the child
        FOREACH c ∈ (children(u_heavy))
            IF (weight(c) == weight(u) AND c ∉ S)
                u_heavy := c
            EN DIF
        END FOREACH

        // subtract weight from parents
        FOREACH p ∈ (parent(u_heavy))
            weight(p) = weight(p) – weight(u_heavy)
            IF (weight(p) < 0)
                weight(p) = 0
            END IF
        END FOREACH
        IF (u_heavy ∉ V)
            V = V ∪ {u_heavy}
        ENDIF

        // remove lighter nodes from the result set
        FOREACH e ∈ (S)
            IF (weight(e) < weight(u_heavy))
                S = S-{e}
            ENDIF
        END FOREACH
        S = S ∪ {u_heavy}

    // do this while there are chunks to be generated
    WHILE (|S| < C OR |S| = |N|)

    // return the result set
    RETURN (S)
END
```

Fig. 6: Algorithm to Generate the Chunking Scheme

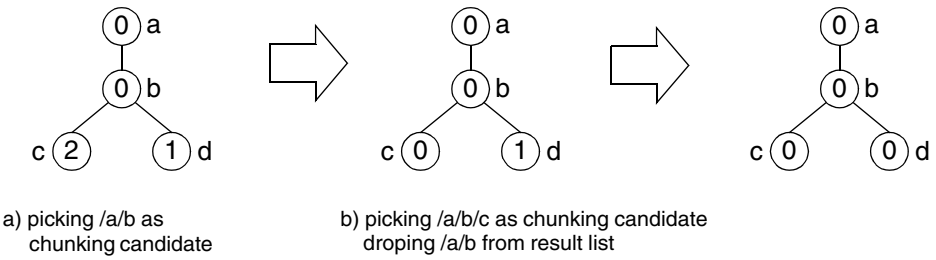


Fig. 7: Sample Query Trees during Developing the Chunking Scheme

duced by the candidate selection process of /a/b. This operation however produces a surprise. The former selected candidate node /a/b now holds a smaller weight than the current candidate node /a/b/c so that the first node does no longer deserve the role of a candidate and is removed from the result list.

The following run of the main DO-WHILE loop produces /a/b/d as the node with the highest weight and the highest depth (compared to /a/b!). Following the same procedure as above, the weight of /a/b and /a are reduced to 0.

Although candidates are potentially removed from the resulting set, the chunking algorithm terminates for any given input, because – in the worst case – all leaf nodes are selected and the weight of the inner nodes are reduced to zero. From a complexity perspective, we obtain $O(n*N)$ with N as the number of nodes in the query tree and n as the number of chunks to create. In the optimal case no candidate node has to be removed from the result list. In the opposite, during the worst case, every node is inserted and removed again on a specific path from the root to a certain leaf node of the query tree. Thus the overall complexity increases to $O(h*n*N)$ with h as the height of the query tree. It may be noted here that the generation of the chunking scheme is performed during the prepare phase and does not count to the time needed to apply a set of XPath filters to an XML document.

Considering XPath Expression with Filters

Considering only simple XPath prefixes as a chunking criterion does not allow the assignment of XPath queries to the corresponding chunks if the queries hold additional filter expressions, so that the chunking scheme would be too strict, i.e. information necessary for evaluating the predicates would be not longer available. Therefore the proposed chunking approach additionally considers filters in adding a set of XPath expressions to identify single XML document nodes to the chunk. For example, if the XPath query is /a/b[c]/d/e and the chunking scheme results in an XML document fragment for /a/b/d, then we add the branch leading to /a/b/c without the remaining subtree, i.e. only the <c>-tagged entry, to the XML fragment.

Figure 8 illustrates the example. The left side shows the XML document fragment holding the part of the original (much larger) document for the chunking prefix /a/b/d. To be able to evaluate the considered XPath query /a/b[c]/d/e during runtime, the chunking scheme is expanded to hold the <c>-tagged node without the subtrees originally rooted by <c>. The

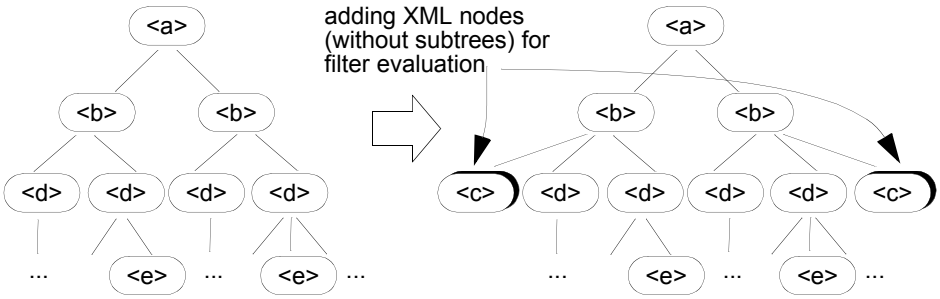


Fig. 8: XML Document Chunks with Nodes Needed to Evaluate Filters in XPath Expressions

capability of resolving filter expressions referring to information from a part of the XML document above the splitting point defined by the chunking criterion requires to retain the structure of the XML document up to the root. This implies that the chunking process for a specific XML document during the filtering phase has a memory requirement to store k XML tag names with k as the maximum of the longest path from the root to any leaf node without any recursion on that path and the depth of the XML document up the first occurrence of a recursion.

Summary of the Process Creating a Chunking Scheme

The transformation steps discussed in the preceding section are a necessary prerequisite to come up with a reasonable chunking scheme shrinking the XML document to a fragment needed to answer a certain set of XPath queries. The chunking scheme is computed based on a weighted query tree with nodes reflecting a single location step. The chunking criteria are iteratively picked so that queries are equally distributed with regard to the generated XML fragments. Locally optimal solutions are discarded and replaced by better solutions during the algorithm. Additionally the chunking scheme considers single nodes of the XML document needed to evaluate XPath queries with filters. This leads to the definition of an XML chunk.

Definition: *XML document chunk*
An XML document chunk C is a tuple $(P, \mathcal{F})$ with P denoting a simple XPath expression identifying the subtrees of an XML document as the base for the XML chunk. The component $\mathcal{F}$ is a set of simple XPath expressions identifying single nodes of the original XML document for evaluating filter expressions.

With the notion of an XML document chunk, we are now able to define a filtering scheme for matching incoming XML documents against a set of registered XPath queries Q split into $n+1$ subsets $Q_1,..., Q_n, Q_{n+1}$ so that the $Q = \bigcup_{i=1}^{n+1} Q_i$.

Definition: *XPath filtering scheme of degree n*
An XPath filtering scheme of degree n consists of a set of $n+1$ tuples with a combination of XML document chunks C_i and a set of XPath query expressions Q_i ($1 \le i \le n+1$), i.e. $\{(C_1,Q_1), ..., (C_n,Q_n), (C_{n+1},Q_{n+1})\}$ so that all XPath expression of Q_i show the same prefix P_i of the corresponding XML document chunk C_i and all filter expressions $\mathcal{F}_i$ occurring in

Q_i can be checked by referring to the nodes specified by the XPath expressions of $\mathcal{F}_i$ for $1 \leq i \leq n$. The potpourri set of XPath query expressions Q_{n+1} can be evaluated referring to chunk C_{n+1}. This potpourri chunk C_{n+1} is either empty (if Q_{n+1} is empty) or corresponds to the original XML document.

5 Performance Evaluations

This section illustrates the core issues of the implementation of the proposed chunking framework together with performance figures comparing runtimes of evaluating multiple XPath queries based on XML document fragments with the original file.

Architecture of the Implementation and Technical Setup

The complete filtering process of XPath expressions is implemented in the context of the ›eXtract‹ project. The eXtract system architecture consists of a collection of different tools written in Java to convert XML documents and/or XPath expressions. Figure 9 gives an overview of the filtering process using eXtract tools.

During the prepare phase, a given set of XPath expressions is cleaned and converted into a standardized format using TRANSTOOL followed by building the query tree and determining the chunking criteria based on the algorithm given in the preceding section (PREPTOOL). The result of the prep tool is the complete filtering scheme with multiple chunk definitions and the associated XPath expressions.

During filtering time, an incoming XML document is given to the stream-oriented CHUNKTOOL which performs a prefiltering step keeping only the parts of the original XML document needed to fulfill the current filtering scheme. Finally, the generated XML fragments are used by the FILTERTOOL to evaluate the XPath queries producing the final result of (in many cases very) small XML documents to be delivered to the subscribers.

The following performance test were carried out using our eXtract implementation, written in Java 1.3.1_03 using the SUN XML Pack Spring 0.2 dev bundle package (http://java.sun.com) and additionally the Universal Java XPath Engine JAXEN 1.0beta8 (http://www.jaxen.org) to evaluate XPath expressions. The tools were running on a WinXP machine with an 800MHz Athlon processor and 384MByte memory.

5.1 Scenario 1: University Organization

The schema of the XML scenario we used to demonstrate the benefit of reducing the size of XML documents before feeding it into the filtering engine is given in appendix A1. The sample XML file counts 130.000 XML tags resulting in 4.85MByte size on disk. As can be seen in the DTD, the scenario holds five different blocks of information (arbitrarily mixed within the XML file).

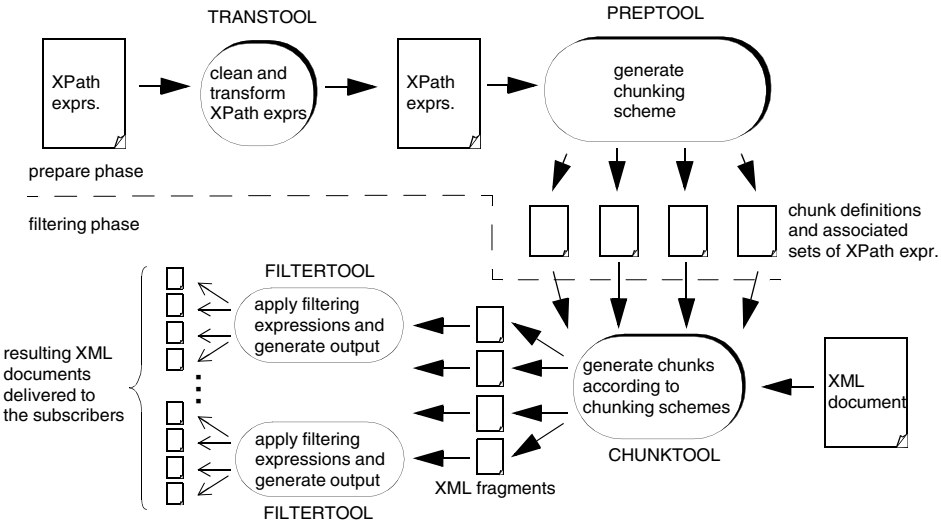


Fig. 9: Base architecture of the eXtract filtering engine implementation

All performance evaluations were carried out using 100, 500, and 2000 registered XPath queries. The query set was synthetically generated so that 30% of the queries exhibit a //-expression and 30% of the queries exhibit an additional filter expression. Figure 10 shows the distribution of the queries referring to one of the five main partitions of the XML document for each scenario separately. Additionally, figure 10 shows the prefixes not selected as real chunks. For example, if we have a chunking degree of 4 (figure 10a), then all students go into the potpourri for the 100 query scenario, the secretaries for the 500 query scenario, and finally professors are making up the potpourri when considering the 2000 query scenario. If the number of real chunks is reduced to 3, then two types of employees are assigned to the potpourri chunk (figure 10b).

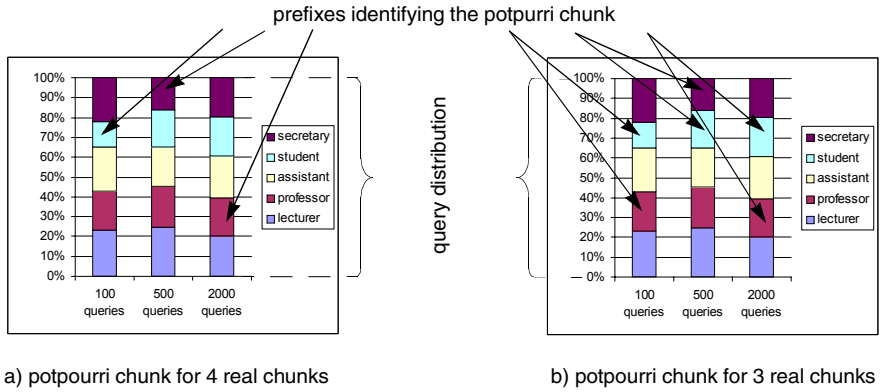


Fig. 10: Query distribution for 100, 500, and 2000 XPath expressions

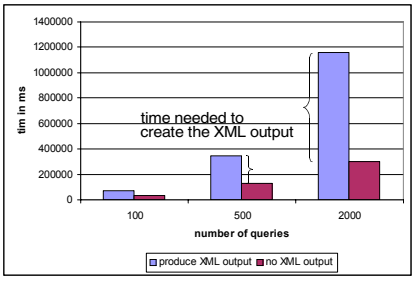
Interpretation of the Performance Measurements

The first diagram of figure 11 shows the different runtimes needed to apply XPath query expressions and to generate the XML output for the subscribers. Although, producing the XML output streams does not influence the results relatively to each other, we omit the result generation when considering the following scenarios to focus on the time needed for the filtering step. Each of the following three scenarios is executed on a filtering scheme with five, four, and finally three real chunks. While the chunking scheme with five real chunks produces an empty potpourri query set, the case with four and three chunks generates a potpourri query set which has to be evaluated with regard to the original XML document.

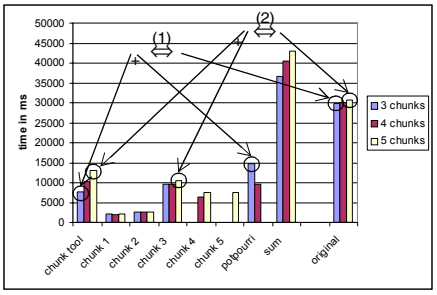
The first component of the diagram of figure 11 denotes the time needed to perform the chunking of the original document according to the given filtering scheme. The numbers for chunk 1 up to chunk 5 together with potpourri denote the time needed to perform the filtering of the associated XPath expressions. The sum component is used to compare the overall filtering time for each chunking scheme with the time needed to perform the filtering based on the original XML document without any modification and optimization.

While the filtering process for 100 queries (figure 11b) based on the original XML document requires less time compared to the sequential execution of the chunked version, the chunking scheme outperforms the original scenario, if we consider a naive parallel environment. Since parallel execution (in theory) would be bound to the longest running filtering step plus the time needed for the initial chunking of the XML document, the filtering process for the 5-chunk scenario would be finished after 10325ms+13059ms for the chunking compared to 30624ms for the sequential filtering method, thus gaining 24% performance speedup ((2) in figure 11b). For the case with 3 chunks, the required potpourri chunk slows down the execution. However, due to a smaller chunking time, the speedup reaches again 25% ((1) in figure 11b).

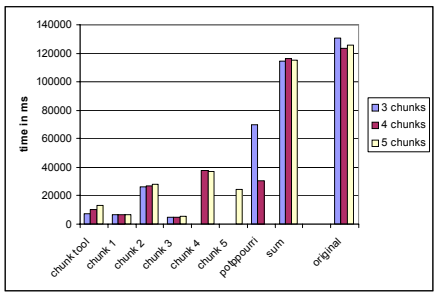
As can be seen in the figure 11c and 11d, the higher the number of XPath expressions to evaluate the higher the speedup gained by the chunking mechanism. For 500 and 2000 queries the sequential execution is already faster than the method based on the complete XML document, so that prefiltering sounds attractive. More detailed, the 2000 query scenario reaches a performance gain of 41% for the scenario with 3 chunks and a potpourri chunk (160981ms + 7491ms compared to 289365ms; (1) in figure 11d). The gain rises to 71% when applying the almost optimal filtering scheme with 5 chunks resulting in an empty potpourri chunk ((2) in figure 11d). In this case, chunk 4 needs 74126ms, resulting in a benefit of 71% when considering an initial chunking period of 12829ms and 298980ms needed to perform 2000 XPath queries based on the original XML document.



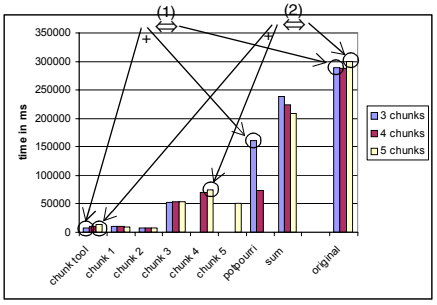
a) time needed to produce XML output



b) evaluating 100 XPath expressions



c) evaluating 500 XPath expressions



d) evaluating 2000 XPath expressions

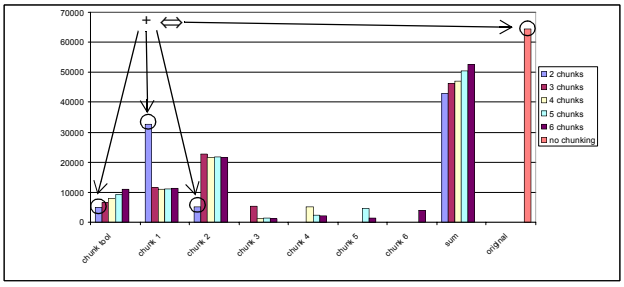
Fig. 11: Performance Measurements for Scenario 1

5.2 Scenario 2: EJB Deployment Descriptor

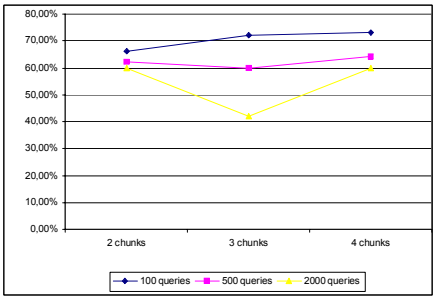
A second scenario demonstrating the feasibility of the proposed chunking approach is based on an EJB deployment description of a large database application project. The DTD is given in appendix A2. The cardinalities of the resulting chunks with the selected XPath prefixes and the query distribution are given in table of appendix A3.

Figure 12a illustrates the result of performance studies for this EJB scenario. For a query set with 100 queries, splitting the original file into two chunks already yields a performance reduction from 64473ms to (5027+(32757+5087))=42871ms for chunking and XPath evaluation with regard to the single chunks. It is interesting to note that in this specific context this splitting scheme seems to be the optimal chunking scheme, because further splits increase the overall time needed to evaluate the XPath queries.

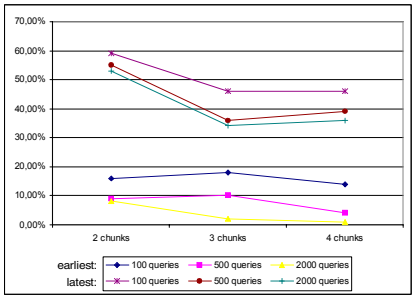
Figure 12b und c show a summary of performance gains. The average runtime needed for a chunking scheme with 2, 3, and 4 chunks is given in figure 12b for query sets with 100, 500, and 2000 XPath expressions. An optimization with only two chunks results in a reduction of the runtime to 60-70% compared to the original case. Since the time needed for the chunking tool and the query execution costs for the individual chunks are summarized



a) 100 XPath queries with a chunking scheme of 2 to 6 chunks



b) ratio of chunk-based compared to regular XPath evaluation



c) earliest and latest result delivery

Fig. 12: Performance Measurements for Scenario 2

and therefore comparable with the execution based on the original XML document, the chunking scheme exhibits another advantage. Individual chunks with the corresponding XPath expressions may be prioritized so that the execution order determines the time of the earliest/latest delivery of the XPath evaluations. Figure 12c holds this information computed by the time needed for the chunking plus the minimum/maximum of the XPath expression evaluation based on the chunks compared to the original runtime. For example, when evaluating 2000 XPath expressions on three chunks, the first query set is ready after 2% of the original query runtime.

6 Summary and Conclusion

Providing an efficient filtering mechanism is the core to build an efficient and large scale information dissemination system. Since XML is the base for exchanging data in loose-coupled information systems, we focus on improving the filtering mechanism for subscriptions specified as XPath expressions evaluated on incoming XML document with known schema. The main idea of our approach is to analyze the registered set of XPath expressions during a prepare phase and generate a filtering scheme to partition the set of queries and splitting the incoming XML document into separate chunks. The size reduction of the

XML document is crucial for applying the filtering mechanism. Comprehensive performance evaluations demonstrate performance gains even in case of a sequential execution of the XPath query sets based on the associated XML chunks. Moreover, the proposed solution might be starting point for priority scheme

The advantage increases when performing the filtering process in parallel based on the customized XML chunks. Since our preprocessing step is independent of the underlying filtering technique, is can be integrated into an existing subscription evaluation system.

References

AAB+98 Altinel, M.; Aksoy, D.; Baby, T.; Franklin, M.; Shapiro, W.; Zdonik, S.: DBIS-Toolkit: Adaptable Middleware For Large Scale Data Delivery. In: *SIGMOD'99*, pp. 544-546

AgCL91 Agrawal, R.; Cochrane, R.; Lindsay, B.: On Maintaining Priorities in a Production Rule System. In: *VLDB'91*, pp. 479-487

AlFr00 Altinel, M.; Franklin, M.J.: Efficient Filtering of XML Documents for Selective Dissemination of Information. In: *VLDB 2000*, pp. 53-63

ASS+99 Aguilera, M.; Strom, R.; Sturman, D.; Astley, M.; Chandra, T.: Matching Events in a Content-Based Subscription System. In: *PODC'99*, pp. 53-61

BaWi01 Babu, S.; Widom, J.: Continous Queries over Data Streams. In: *SIGMOD Record 30(3), 2001*, pp. 109-120

BBC+01 Berglund, A.; Boag, S.; Chamberlin, D.; Fernandez, M.F.; Kay, M.; Robie, J.; Simeon, J.: XML Path Language (XPath) 2.0. Working Draft, Version 2.0, World Wide Web Consortium (W3C), 2001, http://www.w3c.org/TR/xpath20/

BeCr92 Belkin, N.J.; Croft, W.B.: Information Filtering and Information Retrieval: Two Sides of the Same Coin? In: *CACM, 35(12),1992*, pp. 29-38

CDTW00 Chen, J.; DeWitt, D.J.; Tian, F.; Wang Y.: NiagaraCQ: A Scalable Continous Query System for Internet Databases. In: *SIGMOD 2000*, pp. 379-390

CFGR02 Chan, C.-Y.; Felber, P.; Garofalakis, M.N.; Rastogi, R.: Efficient Filtering of XML Documents with XPath Expressions. In: *ICDE 2002*, pp. 235-244

CFR+01 Chamberlin, D.; Florescu, D.; Robie, J.; Simeon, J.; Stedanescu, M. (Hrsg.): XQuery: A Query Language for XML, Working Draft. World Wide Web Consortium (W3C), 2001, http://www.w3c.org/TR/xquery/

ChMS02 Chung, C.-W.; Min, J.K.; Shim, K.: APEX: An Adaptive Path Index for XML Data. In: *SIGMOD 2002*

Fall01 Fallside, D.C.: XML Schema Part 0: Primer. W3C Recommendation, World Wide Web Consortium (W3C), 2001, http://www.w3c.org/TR/xschema-0

FJL+01 Fabret, F.; Jacobsen, H.-A.; Llirbat,F.; Pereira, J.; Ross, K.A.; Shasha, D.: Filtering Algorithms and Implementation for Very Fast Publish/Subscribe. In: *SIGMOD 2001*, pp. 115-126

FoDu92 Foltz, P.W.; Dumais, S.T.: Personalized Information Delivery: An Analysis of Information Filtering Methods. In: *CACM 35(12), 1992*, pp. 51-60

GoWi97 Goldmann, R.; Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Database. In: *VLDB'97*, pp. 436-445

Grus02 Grust, T.: Accelerating XPath Location Steps. In: *SIGMOD 2002*

HCH+99 Hanson, E.N.; Carnes, C.; Huang, L.; Konyala, M.; Noronha, L.; Parthasarathy, S.; Park, J.B.;Vernon, A.: Scalable Trigger Processing. In: *ICDE'99*, pp. 266-275

KrRo95 Krishnamurthy, B.; Rosenblum, D.; Yeast: A General Purpose Event-Action System. In: *TSE 21(10), 1995*, pp. 845-857

LaPa02 Lakshmanan, L.V.S.; Parthasarathy, S:: On Efficient Matching of Streaming XML Documents and Queries. In: *EDBT 2002*, pp. 142-160

LiMo01 Li, Q.; Moon, B.: Indexing and Querying XML Data for Regular Path Expression. In: *VLDB 2001*, pp. 361-370

LiPT99 Liu, L.; Pu, C.; Tang, W.: Continual Queries for Internet Scale Event-Driven Information Delivery. In: *TKDE 11(4), 1999*, pp. 610-628

LPCZ01 Lehner, W.; Pirahesh, H.; Cochrane, R.; Zaharoudakis, M.: fAST Refresh using Mass Query Optimization. In: *ICDE 2001*, pp. 391-398

MiSu99 Milo, T.; Suciu, D.: Index Structures for Path Expressions. In: *ICDT'99*, pp. 277-295

PFJ+01 Pereira, J.; Fabret, F.; Jacobson, H.A.; Llirbat, F.; Shasha, D.: WebFilter: A High-throughput XML-based Publish and Subcribe System. In: *VLDB 2001*, pp. 723-724

RoWo97 Rosenblum, D.S.; Wolf, A.L.: A Design Framework for Internet-Scale Event Observation and Notification. In. *SIGSOFT'97*, pp. 344-360

WHA+00 Wood, L.;Hors, A.L.; Apparao, V.; Byrne, *S.;* Champion, M.; Isaacs, S.; Jacobs, I.; Nicol, G.; Robie, J.; Sutor, R.; Wilson, C.: Document Object Model (DOM) Level 1 Specification (Second Edition).W3C Working Draft. World Wide Web Consortium (W3C), 2000, http://www.w3.org/TR/REC-DOM-Level-1

YaGa95 Yan, T.W.; Garcia-Molina, H.: SIFT - A Tool for Wide Area Information Dissemination. In: *USENIX'95*, pp. 177-186

Appendix A: Description of Sample Scenarios

A1) DTD and Chunk Size of Sample Scenario 1

```
<!ELEMENT db (student+,professor+,assistant+,secretary+,lecturer+) >
<!ELEMENT student (name,email?,url?,link?)>
<!ATTLIST student id ID #REQUIRED>
<!ELEMENT professor (name,email?,url?,link?)>
<!ATTLIST professor id ID #REQUIRED>
<!ELEMENT assistant (name,email?,url?,link?)>
<!ATTLIST assistant id ID #REQUIRED>
<!ELEMENT secretary (name,email?,url?,link?)>
<!ATTLIST secretary id ID #REQUIRED>
<!ELEMENT lecturer (name,email?,url?,link?)>
<!ATTLIST lecturer id ID #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ATTLIST url href CDATA #REQUIRED>
<!ELEMENT link (#PCDATA)>
<!ATTLIST link manager IDREF #IMPLIED subordinates IDREFS #IMPLIED>
```

# of chunks / size in KB	3	4	5
chunk 1	170	170	170
chunk 2	206	206	206
chunk 3	1233	1233	1233
chunk 4		1145	1145
chunk 5			1537
potpurri chunk	4969	4969	0

chunk and potpurri size

A2) DTD of Sample Scenario 2

```
<!ELEMENT assembly-descriptor ( container-transaction+, security-role+, method-permission+ ) >
<!ELEMENT cmp-field ( field-name ) >
<!ELEMENT container-transaction ( method+, trans-attribute ) >
<!ELEMENT ejb-jar ( enterprise-beans, assembly-descriptor ) >
<!ELEMENT ejb-ref ( ejb-ref-name, ejb-ref-type, home, remote, ejb-link ) >
<!ELEMENT enterprise-beans ( message-driven, session+, entity+ ) >
<!ELEMENT entity ( ejb-name, home, remote, ejb-class, persistence-type,
                   prim-key-class, reentrant, cmp-field+, primkey-field? ) >
<!ELEMENT message-driven ( ejb-name, ejb-class, message-selector, transaction-type,
                           acknowledge-mode, message-driven-destination, ejb-ref+ ) >
<!ELEMENT message-driven-destination ( destination-type ) >
<!ELEMENT message-selector EMPTY >
<!ELEMENT method ( ejb-name | method | method-intf | method-name | method-params )* >
<!ELEMENT method-params ( method-param* ) >
<!ELEMENT method-permission ( role-name, method+ ) >
<!ELEMENT run-as ( role-name ) >
<!ELEMENT security-identity ( run-as ) >
<!ELEMENT security-role ( description, role-name ) >
<!ELEMENT session ( ejb-class | ejb-name | ejb-ref | home | remote |
                    security-identity | session-type | transaction-type )* >
```

all other not explicitly mentioned ELEMENTs are of type #PCDATA

A3) Chunk Size and Query Distribution of Sample Scenario 2

number of chunks	relative query distribution	prefix selection for individual chunks	chunk cardinality
Chunk 1	54%	/ejb-jar/assembly-descriptor	3,972
Chunk 2	46%	/ejb-jar/enterprise-beans	681
Chunk 1	19%	/ejb-jar/assembly-descriptor/container-transaction	1,351
Chunk 2	35%	/ejb-jar/assembly-descriptor	3,972
Chunk 3	46%	/ejb-jar/enterprise-beans	681
Chunk 1	19%	/ejb-jar/assembly-descriptor/container-transaction	1,351
Chunk 2	18%	/ejb-jar/assembly-descriptor/method-permission	2,583
Chunk 3	17%	/ejb-jar/assembly-descriptor/security-role	41
Chunk 4	46%	/ejb-jar/enterprise-beans	681
Chunk 1	19%	/ejb-jar/assembly-descriptor/container-transaction	1,351
Chunk 2	18%	/ejb-jar/assembly-descriptor/method-permission	2,583
Chunk 3	17%	/ejb-jar/assembly-descriptor/security-role	41
Chunk 4	16%	/ejb-jar/enterprise-beans/session	162
Chunk 5	30%	/ejb-jar/enterprise-beans	681
Chunk 1	19%	/ejb-jar/assembly-descriptor/container-transaction	1,351
Chunk 2	18%	/ejb-jar/assembly-descriptor/method-permission	2,583
Chunk 3	17%	/ejb-jar/assembly-descriptor/security-role	41
Chunk 4	16%	/ejb-jar/enterprise-beans/session	162
Chunk 5	15%	/ejb-jar/enterprise-beans/message-driven	44
Chunk 6	14%	/ejb-jar/enterprise-beans/entity	497

XML-Archivierung betriebswirtschaftlicher Datenbank-Objekte*

Bernhard Zeller [1] Axel Herbst [2] Alfons Kemper [1]

[1] Universität Passau
94030 Passau, Germany
⟨Nachname⟩@db.fmi.uni-passau.de

[2] SAP AG
69190 Walldorf, Germany
axel.herbst@sap.com

Abstract: Einer der wichtigsten Einsatzbereiche relationaler Datenbanksysteme liegt in der Verwaltung und Auswertung betriebswirtschaftlicher Daten. Insbesondere dienen relationale Datenbanken als Backend für betriebswirtschaftliche Software. In diesem Umfeld ist das Konzept der Archivierung bekannt. Unter Archivierung versteht man dabei das Verschieben der Daten von selten benötigten betriebswirtschaftlichen Objekten aus den OLTP-Datenbanksystemen auf Tertiärspeichersysteme. Dadurch werden das Datenvolumen der Datenbank reduziert, die Leistung des Datenbanksystems erhöht und Kosten gespart. In SAP-Systemen wurde dieses Konzept unter dem Namen *Datenarchivierung* umgesetzt. Wir schlagen in dieser Arbeit vor, die relationalen Datenbanksysteme um einen XML-Archivierungs-Operator zu erweitern. Der XML-Archivierungs-Operator erlaubt es, den gesamten Archivierungsvorgang auf Datenbank-Ebene auszuführen und die Daten als XML-Dokumente abzulegen. Der Operator erhält die Daten eines betriebswirtschaftlichen Objektes in Form von *temporären Tabellen*. Die temporären Tabellen repräsentieren das relationale Schema des Objektes und beinhalten Verweise auf die zugeordneten Tupel der operativen OLTP-Datenbasis. Ein ebenfalls übergebenes *XML-Schema*-Dokument enthält die genaue Definition des Archivobjektes und gibt an, welche Teile des betriebswirtschaftlichen Objektes archiviert werden sollen. Bei der Archivierung stellt das abschließende Löschen der Tupel wegen der vielen benötigten Schreibsperren eine besonders kritische Phase dar. Deshalb wurde hierfür eine effiziente Technik, bei der Datensätze gemäß ihrer physikalischen Anordnung gelöscht werden, in den XML-Archivierungs-Operator integriert.

1 Einleitung

Einer der wichtigsten Einsatzbereiche relationaler Datenbanksysteme liegt in der Verwaltung und Auswertung betriebswirtschaftlicher Daten (*betriebswirtschaftliche Datenbanksysteme*). Die Datenbanksysteme fungieren dabei als Datenspeicher für betriebswirtschaftliche Anwendungssysteme und dienen als Integrationsplattform aller operationalen Daten eines Unternehmens. Trotz des starken Einsatzes in diesem Bereich bieten die heutigen Datenbanksysteme keine oder nur sehr eingeschränkte Möglichkeiten, rudimentäre betriebswirtschaftliche Vorgänge, wie z.B. das Definieren und Sperren betriebswirtschaftlicher Objekte, auf Datenbank-Ebene auszuführen. Diese Defizite führen dazu, dass Hersteller von betriebswirtschaftlicher Standardsoftware typische Datenbankfunktionen, wie die Sperrverwaltung oder die Überwachung komplexer Integritätsbedingungen,

*Diese Arbeit wurde durch die Firma SAP im Rahmen des sog. Terabyte-Projektes gefördert.

auf Applikations-Ebene in ihre Systeme integrierten und die Datenbank nur als reine Speicherschicht verwenden.

Es ist deshalb notwendig, die Schnittstellen der Datenbanken zu erweitern, anzupassen und zu vereinfachen. Ein erster Schritt in diese Richtung sind die XML-Schnittstellen und die objekt-relationalen Erweiterungen, die fast alle Datenbankhersteller mittlerweile anbieten.

Wir schlagen in dieser Arbeit eine Erweiterung relationaler Systeme um einen XML-Archivierungs-Operator vor. Das Konzept der Archivierung ist im Umfeld betriebswirtschaftlich genutzter Datenbanken bekannt und in SAP-Systemen unter dem Namen *Datenarchivierung* umgesetzt [SBB+02]. Die Archivierung ist ein weiteres Beispiel für eine auf Applikations-Ebene implementierte Datenbank-Funktion. Unter Archivierung versteht man dabei das Verschieben der Daten von selten benötigten betriebswirtschaftlichen Objekten aus den produktiven Datenbanksystemen auf kostengünstigere Tertiärspeichersysteme, bei denen CD's oder Bänder als Speichermedien eingesetzt werden. Die Daten sind auch nach der Archivierung noch von der Anwendung aus zugreifbar. Durch das Verschieben der Daten aus der produktiven Datenbank (die in der Praxis in mehreren (System-) Kopien und zusätzlich oft gespiegelt vorliegt) wird das Datenvolumen verkleinert und die Leistung der Datenbank somit erhöht. Zudem sinken die Kosten für das Gesamtsystem, da Tertiärspeicher in der Regel billiger ist als Sekundärspeicher und der Aufwand für die Administration des Systems wesentlich reduziert wird.

Die Hersteller betriebswirtschaftlicher Anwendungen müssen jedes Jahr hohe Ausgaben für die Wartung und Implementierung der Archivierungssoftware tätigen. Der hohe Aufwand rührt daher, dass jede Anwendung ihre eigene Archivierungskomponente implementiert und die Verarbeitung komplexer betriebswirtschaftlicher Objekte schwierig ist, weil auf Datenbankebene die Daten der Objekte auf sehr viele Tabellen verteilt sind (siehe Abbildung 1). Desweiteren muss ein hoher Aufwand betrieben werden, um bei einem Versionswechsel auf Anwendungs- und Systemseite die Lesbarkeit der Archivdaten zu gewährleisten (z.B. bei Schema-Änderungen in der Datenbank oder der Umstellung des Zeichensatzes). Lediglich bei komponentenbasierten Systemen wie SAP R/3 lassen sich diese Kosten durch den Zugriff mehrerer Anwendungen auf eine gemeinsame Archivierungskomponente etwas senken. Im Falle des Systems SAP R/3 ist diese gemeinsam genutzte Archivierungskomponente das *ADK* (Archive Development Kit) [SBB+02, SR97]. Bei der Archivierung liest die Archivierungskomponente die Daten der zu archivierenden betriebswirtschaftlichen Objekte aus der Datenbank, packt sie in (meist nur für die Anwendung lesbare) Dateien und legt diese auf einem dafür vorgesehenen Ablagesystem ab. Anschließend werden die Daten auf Grundlage der zuvor generierten Archivdateien in der produktiven Datenbank gelöscht. Vor allem dieser Löschvorgang kann die Leistung des Datenbanksystems bei sehr vielen zu löschenden Daten stark beeinträchtigen und stellt deshalb ein großes Problem gerade bei der Archivierung großer Datenbankvolumen dar. Dieses Vorgehen war aber aus folgenden Gründen nötig:

- Die Definition der betriebswirtschaftlichen Objekte ist nur auf Anwendungsseite bekannt. Mit der Definition eines betriebswirtschaftlichen Objektes ist dabei vor allem das Wissen gemeint, welche Tabellen in der Datenbank die Daten welches betriebswirtschaftlichen Objektes speichern. Bisher gab es noch keine standardisierten Techniken, mit denen dieses Wissen in die Datenbank transferiert werden konnte.

Mit der Einführung von XML und XML-Schema [XML00] sind aber entsprechende Formate geschaffen worden.

- Die rechtlichen und betriebswirtschaftlichen Regelungen, wann ein Objekt archiviert werden darf und wann nicht, sind derart komplex, dass sie sich nicht mit SQL-Mitteln abbilden lassen. Es sind dazu komplexe Programme auf Anwendungsseite nötig.

- Es gab noch keinen adäquaten Operator auf Datenbank-Ebene, dem auf einfache Weise mitgeteilt werden konnte, wie ein betriebswirtschaftliches Objekt aussieht, welche Objekte zu archivieren sind und der die Daten im produktiven System auf effiziente Weise, nach dem Erzeugen der Archivdateien, löscht.

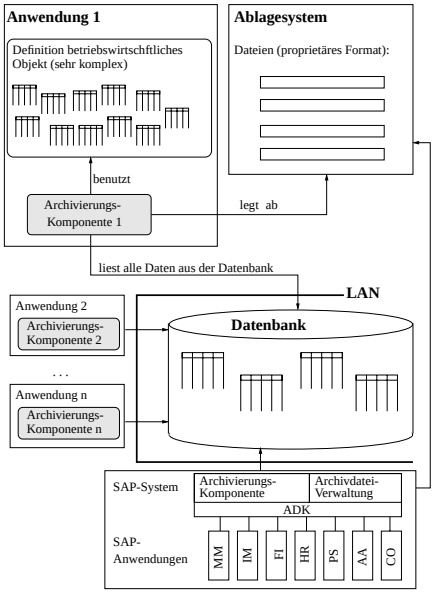


Abbildung 1: Architektur im Bereich der betriebswirtschaftlichen Datenarchivierung

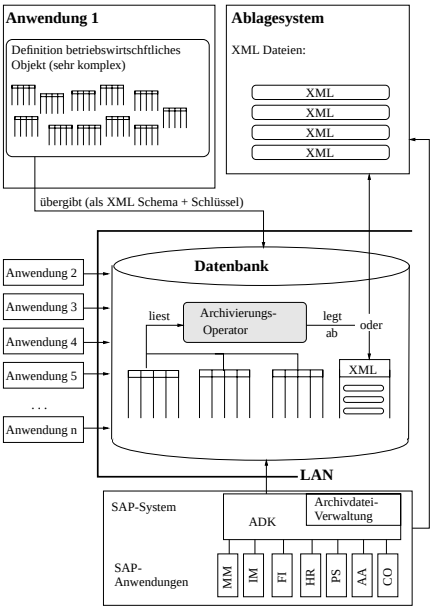


Abbildung 2: Neue Architektur mit XML-Archivierung

Der in dieser Arbeit vorgeschlagene XML-Archivierungs-Operator erlaubt es, im Gegensatz zu den auf Applikations-Ebene implementierten Archivierungskomponenten, den gesamten Archivierungsablauf auf Datenbank-Ebene auszuführen und die Daten als XML-Dokumente abzulegen (siehe Abbildung 2). Dadurch wird der Datenverkehr zwischen Anwendung und Datenbank verringert. Durch die Ablage als XML wird die Portabilität der Daten erhöht und die Implementierung separater Archivierungskomponenten unnötig gemacht. Vor allem die Ablage als XML macht die Daten nicht nur für andere Anwendungen verfügbar, sondern minimiert auch den Aufwand (und damit die Kosten) bei einem Versionswechsel auf Anwendungsseite erheblich.

Desweiteren nutzt der XML-Archivierungs-Operator intelligente Löschalgorithmen, um den Löschvorgang am Ende einer Archivierung zu beschleunigen. Das abschließende Löschen der Tupel ist wegen der vielen benötigten Schreibsperren eine besonders kritische Phase der Archivierung. Die hierfür realisierte effiziente Technik ist eine Variante des von uns in [GKKZ01] vorgestellten *Bulkdelete*-Operators. Wie unsere Messungen zeigen, beschleunigt diese Variante des Bulkdelete-Operators den Löschvorgang um ein Vielfaches (siehe Abschnitt 4).

Die restliche Arbeit ist wie folgt gegliedert: In Abschnitt 2 wird auf verwandte Arbeiten eingegangen. In Abschnitt 3 wird das Konzept der Archivierung genauer erklärt und der neue XML-Archivierungs-Operator vorgestellt. Eine Beschreibung einer Beispiel-Implementierung und die Ergebnisse einiger Messungen sind in Abschnitt 4 zu finden. Abschnitt 5 fasst die vorgestellten Techniken zusammen.

2 Verwandte Arbeiten

Das Konzept eines in die Datenbank integrierten Archivierungs-Operators wurde in [Sch01, Her97, KS98, SBH+98, LS98] erläutert und eine dafür geeignete Erweiterung der SQL-Syntax vorgeschlagen. Im Gegensatz zum XML-Archivierungs-Operator speichern die in diesen Arbeiten vorgeschlagenen SQL-Spracherweiterungen die zu archivierenden Daten in speziellen Archivtabellen und nicht als XML-Dokumente. Außerdem werden die traditionellen Löschmethoden benutzt.

Das Erzeugen von XML-Dokumenten aus relationalen Daten wurde ebenfalls schon detailliert in verschiedenen Projekten untersucht. In [Rys01] sind die XML-Generierungstechniken des Datenbanksystems SQL Server von Microsoft beschrieben. In [CFI+00] wird das Middleware-System XPERANTO, das eine XML-basierte Anfrageschnittstelle zu objekt-relationalen Datenbanken bietet, vorgestellt. In [SSB+01] wurden verschiedene Techniken zum Erzeugen von XML-Dokumenten aus relationalen Daten untersucht. Im System SilkRoute [FTS00] wird eine Kombination aus SQL und XML-QL [DFF+99] verwendet, um das Erzeugen der XML-Dokumente zu steuern. Das Datenbanksystem der Firma Oracle benutzt seit der Version 8i objekt-relationale Typen und Sichten, um die Struktur des zu erzeugenden XML-Dokumentes zu bestimmen [VW02]. Bei der Entwicklung des XML-Archivierungs-Operators lag der Schwerpunkt nicht auf der Untersuchung der XML-Generierung, sondern auf dem verwendeten Löschalgorithmus, der Übergabe der betriebswirtschaftlichen Objekte und dem Fakt, dass die Daten nach der Generierung als XML vorliegen. Wir haben deshalb eine sehr einfache, auf JDOM [JDO] basierende, Art der Generierung angewandt. Prinzipiell können bei der XML-Generierung aber auch alle anderen hier angesprochenen Techniken im XML-Archivierungs-Operator verwendet werden.

In [Moh02] wird ein auf Index Scans basierender Algorithmus zum effizienten Löschen von Tabelleneinträgen vorgestellt, der verwendet werden kann, wenn die zu löschenden Tupel durch eine Bereichsanfrage spezifiziert sind. Der im XML-Archivierungs-Operator verwendete Löschalgorithmus ist eine Variante des in [GKKZ01] vorgestellten Algorithmus. Im Gegensatz zu [Moh02] müssen die zu löschenden Tupel nicht als Bereichsanfrage spezifiziert werden, sondern es kann jede Art von Anfrage benutzt werden.

Eine mögliche Alternative zur Archivierung ist die horizontale Partitionierung der Daten. Auch bei der Partitionierung kann, wie bei der Archivierung, die Wartbarkeit und Leistung der Datenbank erhöht werden. In [Now01] wird gezeigt, welche Partitionierungstechniken in den heute verfügbaren Datenbanksystemen integriert sind und wie diese in der Archivierung eingesetzt werden können [Now99]. In [KN99] wurden Techniken untersucht, um Datenbanken mittels Partitionierung von gewissen Datenbeständen zu maskieren. In [ZK02] wurde untersucht, inwiefern Partitionierung im SAP-Umfeld einsetzbar ist.

Im Rahmen der Entwicklung des XML-Archivierungs-Operators entwickeln wir auch einen XML-Archiv-Browser, der es erlaubt, ein Ablagesystem über Protokolle wie WebDAV oder SOAP [BEK+00] anzusprechen und die abgelegten XML-Archivdokumente mittels benutzerdefinierter Stylesheets darzustellen.

3 Der XML-Archivierungs-Operator

Um es den Programmierern betriebswirtschaftlicher Software zu ermöglichen, mehr Datenbankfunktionen zu nutzen, müssen auf Datenbankseite bessere und angepasstere Schnittstellen und Operatoren zur Verfügung gestellt werden. Der in dieser Arbeit vorgeschlagene XML-Archivierungs-Operator erlaubt es, die Archivierung von betriebswirtschaftlichen Objekten auf Datenbank-Ebene auszuführen. Der XML-Archivierungs-Operator versteht komplexe Objektdefinitionen, kann die Daten in einem geeigneten Austauschformat (XML) ablegen (*Archivdaten-Generierung*) und löscht die Daten anschließend effizient aus der produktiven Datenbasis (*Löschvorgang*). Möglich ist dies zum einen durch die Art und Weise, wie dem XML-Archivierungs-Operator die Daten über die zu archivierenden betriebswirtschaftlichen Objekte übergeben werden und zum anderen durch einen besonderen Algorithmus zum Löschen der Daten aus der produktiven Datenbasis. Diese beiden Techniken bilden das Herzstück des XML-Archivierungs-Operators. Bevor genauer auf diese Techniken eingegangen wird, soll zunächst das Konzept der Archivierung vertieft werden.

3.1 Archivierung betriebswirtschaftlicher Datenbank-Objekte

Der wichtigste Grund für eine Archivierung sind Performance-Probleme der Datenbank, die aus zu groß gewordenen Tabellen resultieren. Im SAP-Umfeld gibt es bereits Systeme, deren Datenbasen eine Größe von mehreren Terabyte erreicht haben und kontinuierlich weiter wachsen. Ein Problem sind z.B. die Beeinträchtigungen beim Pflegen der Datenbasis. Arbeiten, wie das Aufbauen von Statistiken und Indexen, dauern bei großen Tabellen sehr lange und können den dafür vorgesehenen (zeitlichen) Rahmen sprengen. Dies wiederum kann dazu führen, dass diese Arbeiten gar nicht mehr durchführbar sind und sich die Leistung des Systems deshalb immer mehr verschlechtert. Die Archivierung kann solche Probleme lösen, indem sie die Daten nicht mehr oder nur noch selten benötigter betriebswirtschaftlicher Objekte von den Tabellen der produktiven Datenbasis auf Tertiärspeichersysteme verschiebt und damit die Tabellen im produktiven Datenbanksystem wieder verkleinert. Beim Archivieren in betriebswirtschaftlichen Anwendungen sollen aber nicht einzelne Tabelleneinträge archiviert werden, sondern immer die gesamten Daten eines betriebswirtschaftlichen Objektes, also z.B. eines Beleges. Dadurch sind

die archivierten Daten in sich konsistent und können grundsätzlich auch außerhalb des Datenbanksystems weiter bearbeitet werden.

Wie bereits erwähnt, werden nur die betriebswirtschaftlichen Objekte archiviert, die voraussichtlich nur noch selten (oder gar nicht mehr) benötigt werden. Dies kann z.B. der Fall sein, wenn die Daten aus früheren Jahren stammen und deshalb veraltet sind oder die Objekte aus betriebswirtschaftlicher Sicht abgeschlossen sind (z.B. ausgeglichene Finanzbuchhaltungsbelege).

Das Archivieren betriebswirtschaftlicher Objekte geschieht in 2 Schritten:

1. Zuerst werden die Daten der zu archivierenden betriebswirtschaftlichen Objekte aus der Datenbank gelesen und in speziellen (komprimierten) Dateien - sog. *Archivdateien* - gespeichert (**Archivdaten-Generierung**).

2. Anschließend werden die Daten auf Grundlage der zuvor generierten Archivdateien in der produktiven Datenbank gelöscht (**Löschvorgang**).

Damit die Anwendung auf die Archivdateien zugreifen kann, werden sie von einer speziellen Applikation verwaltet. Diese Applikation (das sog. *Ablagesystem*) funktioniert vereinfacht gesagt wie ein Dokumenten Management System. Die archivierende Anwendung übergibt die Archivdateien und einen entsprechenden Schlüssel an das Archivsystem und das Archivsystem liefert auf Anfrage die eingestellte Archivdatei zurück.

Die in diesem Abschnitt vorgestellte Archivierung wurde bisher immer auf Applikations-Ebene implementiert. Der in den folgenden Abschnitten beschriebene XML-Archivierungs-Operator arbeitet dagegen auf Datenbank-Ebene.

3.2 Die Aufruf-Syntax des XML-Archivierungs-Operators

In diesem Abschnitt wird gezeigt, wie dem XML-Archivierungs-Operator die Definition des betriebswirtschaftlichen Objektes übergeben wird und wie der Operator erfährt, welche Objekte zu archivieren sind.

Welche Tabellen in der Datenbank die Daten welches betriebswirtschaftlichen Objektes speichern, ist, wie bereits erwähnt, nur auf Anwendungsseite bekannt. Desweiteren kann nur die Anwendung entscheiden, welche betriebswirtschaftlichen Objekte archiviert werden können. Die Anwendung trifft diese Entscheidung aufgrund juristischer und betriebswirtschaftlicher Regeln, die in Form von Programmen in der Anwendung enthalten sind. Dem XML-Archivierungs-Operator werden diese Daten mittels zweier Parameter übergeben (siehe Abbildung 3): Die Definition des betriebswirtschaftlichen Objektes wird mittels eines *XML-Schema-Dokumentes* übergeben (Parameter **schema**) und das Wissen, welche Objekte zu archivieren sind, mittels spezieller *temporärer Tabellen* (**from**-Klausel).

Die Parameter sowie die restlichen Klauseln der Aufruf-Syntax des XML-Archivierungs-Operators werden in den folgenden Abschnitten genauer erklärt.

3.2.1 Der Parameter *schema*

Mit Hilfe des Parameters **schema** wird dem XML-Archivierungs-Operator mitgeteilt, wie das betriebswirtschaftliche Objekt als XML-Dokument abzulegen ist. In dem übergebenen

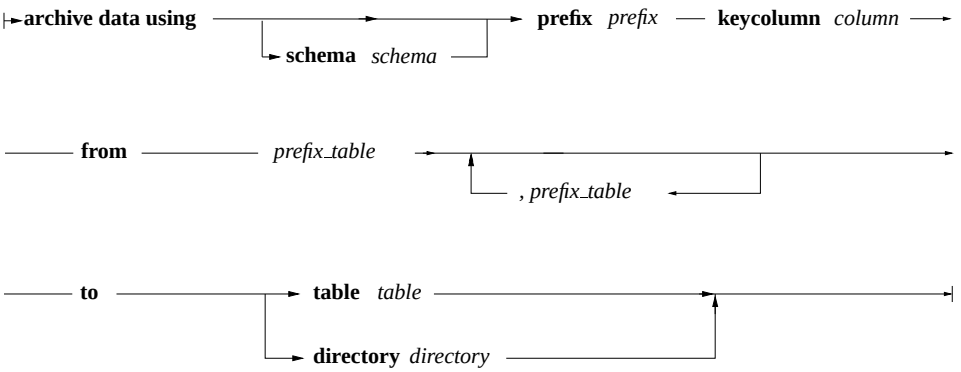


Abbildung 3: Aufrufsyntax des XML-Archivierungs-Operators

XML-Schema ist angegeben, welche Tabellen zu dem betreffenden betriebswirtschaftlichen Objekt gehören, welche Spalten dieser Tabellen relevant sind und wie diese Daten in das zu erzeugende XML-Dokument eingebunden werden sollen. Das übergebene XML-Schema muss dazu den im Folgenden beschriebenen Bedingungen genügen, um korrekt vom Operator bearbeitet werden zu können.

Tabellendaten werden mittels *Annotationen* in das XML-Schema eingebunden. Annotationen sind laut XML-Schema-Spezifikation vorgesehen, um Applikationen Informationen zukommen zu lassen, wie das jeweilige Schema zu verarbeiten ist [XML00]. Im XML-Schema muss dazu an der Stelle, wo die Tabellendaten integriert werden sollen, folgende Annotation eingefügt werden:

```
<xsd:annotation>
   <xsd:appinfo>
      sql:[Schema].[Tabelle].[Spalte]
   </xsd:appinfo>
</xsd:annotation>
```

Bei der XML-Generierung wird dann an dieser Stelle ein XML-Element erzeugt, das den Attributwert eines Tupels enthält. Der XML-Archivierungs-Operator wandelt dabei die Datenbank-Datentypen in passende XML-Datentypen um. Die XML-Datentypen sind mittels sog. *SimpleType*-Definitionen vordefiniert und brauchen nur noch benutzt zu werden. Der von uns implementierte Prototyp (siehe Abschnitt 4) unterstützt z.B. alle im SAP R/3 System verwendeten Datentypen.

Die XML-Elemente aller Attribute eines Tupels ergeben ein in sich konsistentes XML-Fragment. Werden mehrere Tupel einer Tabelle archiviert, so werden mehrere dieser XML-Fragmente hintereinander erzeugt. Abbildung 4 zeigt einen entsprechenden Teil eines XML-Schemas, die dazugehörige Tabelle und die daraus erzeugten XML-Fragmente. Aus Gründen der Übersichtlichkeit ist die Darstellung vereinfacht.

Die Speicherung in XML bietet durch geeignete Wahl der Element-Namen (mnemonische Namen) die Möglichkeit, zusätzliche Information abzuspeichern (z.B. Spalte ANZ speichern mit Element-Namen <Menge>). Die Eindeutigkeit der Element-Namen für die Spaltendefinitionen innerhalb einer Tabelle muss dabei gewährleistet sein.

```
...
<element name="Position"
     type="bestellposition" />
...
<complexType name="bestellposition">
  <sequence>
    <element name="BID" type="NUMBER">
      <annotation><appinfo>
         sql:SAPR3.Bestellposition.BID
      </appinfo></annotation>
    </element>
    <element name="Zeile" type="NUMBER">
      <annotation><appinfo>
         sql:SAPR3.Bestellposition.Zeile
      </appinfo></annotation>
    </element>
    <element name="Ware" type="CHAR">
      <annotation><appinfo>
         sql:SAPR3.Bestellposition.Ware
      </appinfo></annotation>
    </element>
    <element name="Menge" type="NUMBER">
      <annotation><appinfo>
         sql:SAPR3.Bestellposition.Anz
      </appinfo></annotation>
    </element>
    <element name="Preis" type="NUMBER">
      <annotation><appinfo>
         sql:SAPR3.Bestellposition.Preis
      </appinfo></annotation>
    </element>
  </sequence>
</complexType>
```

```
...
<Position>
  <BID>1</BID>
  <Zeile>1</Zeile>
  <Ware>Fernseher</Ware>
  <Menge>1.0</Menge>
  <Preis>1000.0</Preis>
</Position>
<Position>
  <BID>1</BID>
  <Zeile>2</Zeile>
  <Ware>Videorecorder</Ware>
  <Menge>1.0</Menge>
  <Preis>200.0</Preis>
</Position>
...
```

Bestellposition				
BID	**Zeile**	Ware	Anz.	Preis
1	1	Fernseher	1	1000
1	2	Videorecorder	1	200
2	1	Radio	1	57
3	1	1,5 V Batterie	10	0,10
3	2	Lautsprecher	2	23
3	3	CD's	2	30
...	...	...	...	...

Abbildung 4: XML-Schema Ausschnitt mit erzeugten XML-Fragmenten

Durch Zusammenfügen der einzelnen Bausteine für die Tupel eines betriebswirtschaftlichen Objektes ergibt sich ein XML-Schema für dieses Objekt. Dabei ist es möglich, durch geeignete Schachtelung zusätzliche Information innerhalb des erzeugten XML-Dokumentes zu speichern. So können z.B. Fremdschlüsselbeziehungen durch geeignetes Schachteln der Tabellendaten dargestellt werden (siehe hierzu auch das Beispiel in Abschnitt 3.3).

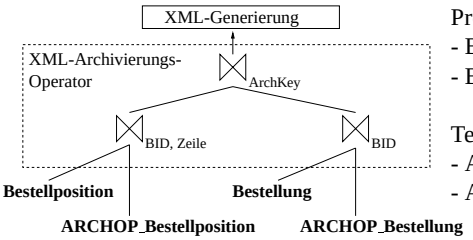
Wird kein Schema angegeben, so erzeugt der Archiv-Operator selbst ein Default-XML-Schema. Er untersucht dazu die Schemata der zu archivierenden Tabellen. Allerdings geht der Operator dabei davon aus, dass alle Attribute dieser Tabellen archiviert werden sollen. Eine Einschränkung auf bestimmte Attribute ist hier nicht möglich. Das erzeugte XML-Schema ist außerdem sehr einfach gehalten und beinhaltet keine zusätzlichen Informationen wie z.B. Fremdschlüsselbeziehungen.

3.2.2 Die *from*-Klausel

Mittels der **from**-Klausel wird dem XML-Archivierungs-Operator mitgeteilt, welche betriebswirtschaftlichen Objekte zu archivieren sind. Dies geschieht durch die Angabe der zu verwendenden *temporären Tabellen* in der **from**-Klausel. Jeder produktiven Tabelle entspricht dabei genau eine temporäre Tabelle. Die Verbindung zwischen temporärer Tabelle und produktiver Tabelle wird über den Namen hergestellt: Der Name der temporären Tabelle ist zusammengesetzt aus dem im Parameter **prefix** angegebenen Präfix und dem Namen der entsprechenden produktiven Tabelle. Die temporären Tabellen bilden das Bin-

deglied, mit dessen Hilfe die Daten eines betriebswirtschaftlichen Objektes identifiziert werden können. Dieses Bindeglied ist nötig, da in der produktiven Datenbasis oft keine direkte Verbindung zwischen den Tabellen eines betriebswirtschaftlichen Objektes (z.B. mittels Fremdschlüssel-Beziehungen) hergestellt werden kann. Die meisten betriebswirtschaftlichen Anwendungen wurden in den letzten Jahren ständig erweitert und nicht alle diese Erweiterungen konnten durch Änderungen im Datenmodell korrekt abgebildet werden. Deshalb sind die Tabellen eines betriebswirtschaftlichen Objektes auf Datenbank-Ebene nicht immer mittels eines Verbundes verknüpfbar. Stattdessen wird das in den Anwendungen vorhandene, zusätzliche Wissen genutzt, um die Tabellen auf Anwendungs-Ebene zu verknüpfen.

Um es trotzdem möglich zu machen, die Daten der produktiven Tabellen zu verbinden, enthalten die temporären Tabellen zum einen eine Spalte mit einem eindeutigen Objektschlüssel und zum anderen Spalten mit den Schlüsselattributen der jeweiligen produktiven Tabelle. Die Spalte mit den Objektschlüsseln trägt den mittels des Parameters **keycolumn** angegebenen Namen. Die Spalten mit den Schlüsselwerten der produktiven Tabelleneinträge tragen den selben Namen wie die entsprechenden Spalten in den produktiven Tabellen. Durch den eindeutigen Objektschlüssel können nun die temporären Tabellen mittels eines natürlichen Verbundes verknüpft werden. Da die temporären Tabellen neben dem Objektschlüssel auch noch die Schlüsselwerte der Einträge der produktiven Tabellen enthalten, können nun auch die produktiven Tabellen verknüpft werden. Für die Beispiel-Tabellen aus Abschnitt 3.3 ist diese Verknüpfung der temporären mit den produktiven Tabellen in Abbildung 5 dargestellt[1].



Produktive Tabellen:
- Bestellung (**BID**, Kunde, Datum, Summe)
- Bestellposition (**BID**, **Zeile**, Ware, Anz, Preis)

Temporäre Tabellen:
- ARCHOP_Bestellung (ArchKey,BID)
- ARCHOP_Bestellposition (ArchKey, BID, Zeile)

Abbildung 5: Verknüpfung der produktiven Tabellen mit Hilfe der temporären Tabellen

Sind die Tabellen entsprechend verknüpft, können die Daten der betriebswirtschaftlichen Objekte ausgelesen und archiviert werden. Die **to**-Klausel gibt dabei an, wohin die Daten gespeichert werden sollen. Ist in der **to**-Klausel eine Tabelle angegeben, so darf diese Tabelle nur 2 Spalten enthalten. In der ersten Spalte werden die Objektschlüssel abgelegt (um die XML-Dokumente eindeutig identifizieren zu können) und in der 2. Spalte die XML-Dokumente. Der Typ der 2. Spalte muss es erlauben, in einer Zeile ein komplettes XML-Dokument zu speichern. Die meisten Datenbankhersteller bieten für solche Zwecke spezielle XML-Datentypen an, aber auch ein CLOB (Character Large Object) kann verwendet werden. Wird das Dateisystem als Ablageort gewählt, wird für jedes Objekt eine

[1] Wegen der Einfachheit des Beispiels wäre hier ein direkter Join der produktiven Tabellen möglich. Dies ist aber bei komplexeren Objekten in der Regel nicht der Fall

Datei erzeugt. Der Dateiname enthält dabei den Objektschlüssel des enthaltenen Objektes und dessen Typ (z.B. ”Beleg4711.xml”).

Im folgenden Abschnitt wird der Ablauf einer Archivierung nochmals genau erläutert.

3.3 Archivieren mit dem XML-Archivierungs-Operator

Beim Einsatz des XML-Archivierungs-Operators läuft eine Archivierung wie folgt ab:

1. Es werden von der Anwendung die temporären Tabellen erzeugt (Gemäß den Vorgaben durch die Definition des betriebswirtschaftlichen Objektes).
2. Die temporären Tabellen werden mit den Schlüsselwerten der Tabelleneinträge des betriebswirtschaftlichen Objektes gefüllt.
3. Der XML-Archivierungs-Operator wird aufgerufen. Dieser verarbeitet die Daten wie folgt:
   (a) Die temporären Tabellen werden komplett gesperrt. Dies ist notwendig, um die Konsistenz der erzeugten XML-Daten zu gewährleisten. Dieses Vorgehen stellt aber keine Problem dar, da die temporären Tabellen nur vom XML-Archivierungs-Operator verwendet werden.
   (b) Der Operator sperrt die Daten aus den produktiven Tabellen mittels einer Lesesperre. Welche Daten zu sperren sind, ist in den temporären Tabellen festgelegt. Anschließend werden die Daten gelesen.
   (c) Die Daten und das mittels des Parameters **schema** übergebene XML-Schema werden verknüpft, und es werden die XML-Dokumente erzeugt.
   (d) Die XML-Dokumente werden an dem in der **to**-Klausel angegebenen Ort abgelegt.
   (e) Die Lesesperren werden in Schreibsperren umgewandelt, und die Daten werden in den produktiven Tabellen gelöscht.
   (f) Der Operator gibt die Sperren frei.
4. Nun können die Daten in den temporären Tabellen gelöscht werden und eventuell die XML-Dokumente weiterverarbeitet werden.

Diese Vorgehensweise soll an folgendem Beispiel verdeutlicht werden: Wir gehen von einem Elektronikfachgeschäft aus. Aus der Datenbasis des Unternehmens sollen Belege archiviert werden. Die Datenbasis aus Abbildung 7 soll dabei als Grundlage dienen (Schlüsselfelder sind fett gedruckt).

Es sollen nun die beiden ersten Bestellungen archiviert werden. Dies geschieht mit folgendem Befehl:

```
archive data using schema /home/zeller/Beleg.xsd
                   prefix ARCHOP_
                   keycolumn ArchKey
from          ARCHOP_Bestellung, ARCHOP_Bestellposition
to            table BelegArchiv
```

Die Archivierung wird nun gemäß des oben dargestellten Algorithmus durchgeführt (siehe Abbildung 6). Bei der Archivierung werden dann die in Abbildung 9 dargestellten XML-Dokumente erzeugt. Diese XML-Dokumente enthalten die Daten der beiden betriebswirtschaftlichen Objekte. Das bei der Archivierung verwendete Schema *Beleg.xsd* ist in Abbildung 8 dargestellt. Sind die XML-Dokumente erzeugt, werden die Daten in den produktiven Tabellen gelöscht. Der dabei verwendete Löschalgorithmus wird im folgenden Abschnitt erklärt.

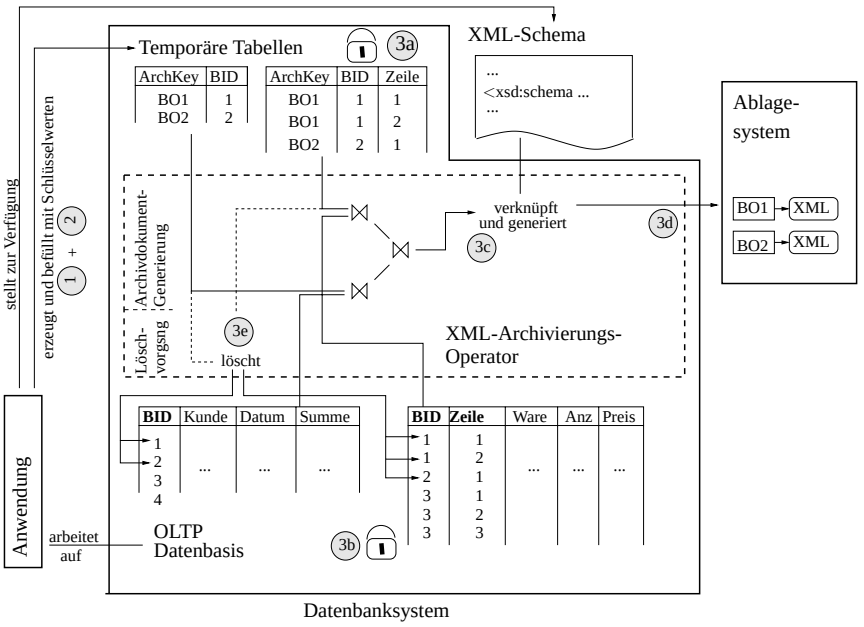


Abbildung 6: Ablauf einer Archivierung gemäß Algorithmus

Bestellung			
BID	Kunde	Datum	Summe
1	4711	3.5.2002	1200
2	5678	3.5.2002	57
3	3456	4.5.2002	107
4	5678	4.5.2002	10
...	...	...	...

Bestellposition				
BID	**Zeile**	Ware	Anz.	Preis
1	1	Fernseher	1	1000
1	2	Videorecorder	1	200
2	1	Radio	1	57
3	1	1,5 V Batterie	10	0,10
3	2	Lautsprecher	2	23
3	3	CD's	2	30
...	...	...	...	...

Abbildung 7: Beispiel-Tabellen eines Elektronikfachgeschäftes

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.sap.com/archive/axml/document" elementFormDefault="qualified"
targetNamespace="http://www.sap.com/archive/axml/document">
  <xsd:element name="Beleg" type="ItemListType" />
  <xsd:complexType name="ItemListType">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="BelegKopf" type="Bestellung" />
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="BelegPositionen">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="Position" type="Bestellposition" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Bestellung">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" name="BID" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellung.BID</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="KundenNummer" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellung.Kunde</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="BestellDatum" type="CHAR">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellung.Datum</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="GesamtSummeInEuro" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellung.Summe</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Bestellposition">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" name="BID" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellposition.BID</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="Zeile" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellposition.Zeile</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="Ware" type="CHAR">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellposition.Ware</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="Menge" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellposition.Anz</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="Preis" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellposition.Preis</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="NUMBER">
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
  <xsd:simpleType name="CHAR">
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
</xsd:schema>
```

Abbildung 8: Beleg.xsd: XML-Schema für die Beispiel-Belege

```
Dokument BO1:
=============

<?xml version="1.0" encoding="UTF-8"?>
<Beleg xmlns="http://www.sap.com/archive/axml/document"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.sap.com/archive/axml/document schema.xsd">
  <BelegKopf>
    <BID>1</BID>
    <KundenNummer>4711</KundenNummer>
    <BestellDatum>20020503</BestellDatum>
    <GesamtSummeInEuro>1200.0</GesamtSummeInEuro>
  </BelegKopf>
  <BelegPositionen>
    <Position>
      <BID>1</BID>
      <Zeile>1</Zeile>
      <Ware>Fernseher</Ware>
      <Menge>1.0</Menge>
      <Preis>1000.0</Preis>
    </Position>
    <Position>
      <BID>1</BID>
      <Zeile>2</Zeile>
      <Ware>Videorecorder</Ware>
      <Menge>1.0</Menge>
      <Preis>200.0</Preis>
    </Position>
  </BelegPositionen>
</Beleg>

Dokument BO2:
=============

<?xml version="1.0" encoding="UTF-8"?>
<Beleg xmlns="http://www.sap.com/archive/axml/document"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.sap.com/archive/axml/document schema.xsd">
  <BelegKopf>
    <BID>2</BID>
    <KundenNummer>4711.0</KundenNummer>
    <BestellDatum>20020503</BestellDatum>
    <GesamtSummeInEuro>57.0</GesamtSummeInEuro>
  </BelegKopf>
  <BelegPositionen>
    <Position>
      <BID>2</BID>
      <Zeile>1</Zeile>
      <Ware>Radio</Ware>
      <Menge>1.0</Menge>
      <Preis>57.0</Preis>
    </Position>
  </BelegPositionen>
</Beleg>
```

Abbildung 9: Erzeugtes XML-Dokument eines Beleg-Objektes

3.4 Der Löschalgorithmus des XML-Archivierungs-Operators

Das massenhafte Löschen von Daten in relationalen Datenbanksystemen kann zu erheblichen Performance-Problemen führen, da die traditionell verwendete Löschmethode die Zugriffe auf den Hintergrundspeicher nicht optimiert. Bei der traditionellen Löschmethode werden die Einträge in den Tabellen einzeln und separat voneinander gelöscht. Dadurch kommt es zu vielen zufälligen Zugriffen auf den Hintergrundspeicher (Random IO), die wiederum wesentlich mehr Zeit in Anspruch nehmen als ein sequenzieller Zugriff (Sequential IO). Wie unsere Messungen in einem kommerziellen Datenbanksystem zeigen, dauert das Löschen von 15% der Einträge aus einer 500 MB großen Tabelle mit 1.000.000 Einträge bereits fast 3 Stunden (siehe Abbildung 10).

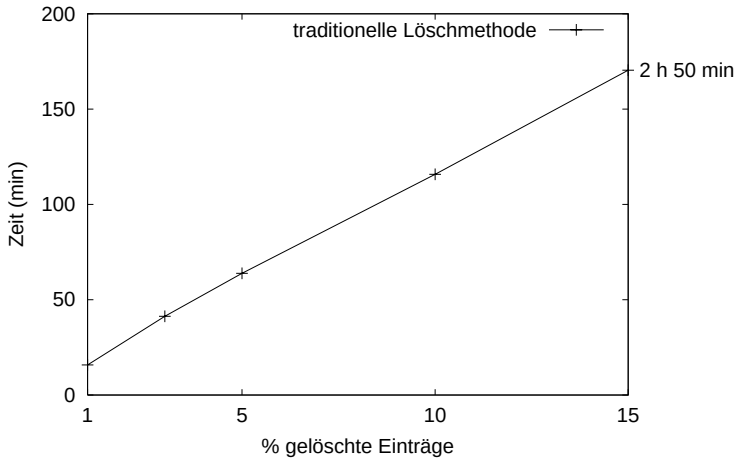


Abbildung 10: Löschen aus einer 500 MB großen Tabelle in einem kommerziellen Datenbanksystem

Wir haben deshalb in [GKKZ01] einen neuen Operator für Massenlöschungen entwickelt: den *Bulkdelete*-Operator. Dieser neue Bulkdelete-Operator vermeidet den zufälligen Zugriff auf den Hintergrundspeicher und erzwingt stattdessen einen sequenziellen Zugriff. Dies geschieht durch geschicktes Ändern der Löschreihenfolge, je nach zu bearbeitender Datenbankstruktur. So wird beim Löschen von Tabelleneinträgen in der Reihenfolge der physikalischen Speicheradresse (RID) gelöscht, und bei B-Baum-Indexen werden die Zugriffe bzgl. der indizierten Werte sortiert. Abbildung 11 zeigt einen möglichen Ausführungsplan für einen solchen Bulkdelete-Löschvorgang. Ein Pfeil neben dem Join-Symbol bedeutet dabei, dass in der Eingaberelation auf der Seite des Pfeiles während des Joinvorgangs auch gleichzeitig Einträge gelöscht werden. Die Menge *D* gibt an, welche Einträge in der Tabelle T gelöscht werden sollen (Delete-Set).

In [GKKZ01] wurden noch weitere Auswertungs- und Optimierungsmöglichkeiten untersucht und auch Messungen mit einem Prototypen durchgeführt. Diese Messungen zeigen, dass der Bulkdelete-Operator um ein vielfaches schneller arbeitet als die herkömmlichen Löschalgorithmen. Der Bulkdelete-Operator ist jedoch in keinem kommerziellen Datenbanksystem verfügbar. Um dessen Kernidee dennoch für den XML Operator zu nutzen,

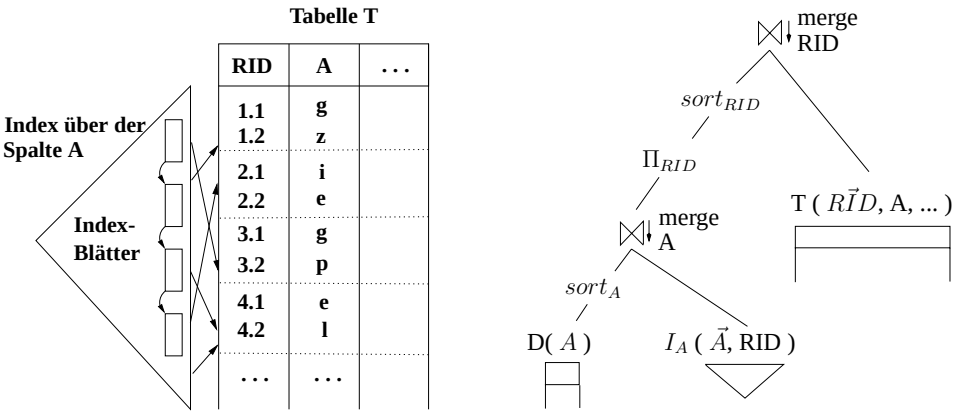


Abbildung 11: Möglicher Auswertungsplan eines Bulkdelete-Operators

haben wir eine leicht zu implementierende *light*-Variante des Bulkdelete-Operators realisiert. Auch die light-Variante erzwingt einen sequenziellen Zugriff durch das Verändern der Löschreihenfolge, allerdings nur beim Zugriff auf die Tabelle. Beim Bearbeiten der Indexe wird auch hier zufällig auf den Hintergrundspeicher zugegriffen.

Die genaue Funktionsweise dieser light-Variante soll an folgendem Beispiel verdeutlicht werden. Gegeben sei eine Tabelle *T*. *T* enthält eine Spalte A, über die ein Index I_A definiert ist (siehe Abbildung 12). Außerdem ist zu jedem Tabelleneintrag die physikalische Speicheradresse abrufbar[2] (hier durch die Spalte *RID* dargestellt). Ein Benutzer möchte nun mehrere Einträge aus *T* löschen und speichert dazu die A-Werte dieser Einträge in einer Tabelle *D* ab (siehe ebenfalls Abbildung 12). Um nun einen sequenziellen Zugriff

T			
RID	...	A	...
001	...	4	...
002	...	7	...
003	...	5	...
004	...	9	...
005	...	3	...
006	...	5	...

D
A
3
7
5

Abbildung 12: Beispieltabellen zur Verdeutlichung der Arbeitsweise der light-Variante

auf *T* zu erzwingen, werden die Einträge der Tabelle *D* bzgl. der physikalischen Adressen der korrespondierenden Einträge aus *T* sortiert. Anschließend werden die Einträge in *T* gelöscht. Dies geschieht mit folgendem SQL-Aufruf:

```
delete from T where T.A in
      (select T.A
       from   D, T
       where  D.A = T.A
       order by T.RID)
```

[2]Die meisten Datenbankhersteller bieten eine entsprechende Funktionalität an. Im nächsten Absatz wird aufgezeigt, welche Tuningmaßnahmen möglich sind, wenn eine entsprechende Funktionalität fehlt.

Dabei ist zu beachten, dass der Join $T \bowtie D$ ausgeführt werden kann, ohne die Tupel aus *T* wirklich zu lesen. Stattdessen kann der Index I_A benutzt werden, um die entsprechenden (T.A, RID) Paare zu finden. Dadurch kann die Unteranfrage sehr effizient beantwortet werden. Ist der Index I_A als B-Baum realisiert und liegt eine Ballung (Clusterung) der Daten in der Tabelle *T* bzgl. der Spalte A vor, so werden durch die light-Variante auch die Blätter des Indexes I_A sequenziell bearbeitet. Sollte kein geeigneter Index existieren (d.h. es existiert kein Index, der ein in *D* gegebenes Attribut indiziert), so kann die light-Variante nicht verwendet werden. In diesem Fall muss aber ohnehin die ganze Tabelle durchsucht werden, da es ohne einen geeigneten Index keine andere Möglichkeit gibt, die zu den Einträgen in *D* korrespondierenden Einträge in *T* zu finden.

Sollte es desweiteren nicht möglich sein, die physikalischen Speicheradresse eines Tabelleneintrages herauszufinden und werden B-Bäume als Indexstrukturen verwendet, so müssen die Einträge in der Tabelle *D* gemäß ihren Attributwerten sortiert werden. Dadurch werden zumindest die Blätter des Indexes I_A sequenziell gelesen. Da die Daten in betriebswirtschaftlichen Datenbanksystemen wegen der Verwendung von künstlichen, aufsteigend nummerierten Schlüsselwerten oft auch sehr gut bzgl. der indizierten Tabellenspalten geballt sind, wird auf diese Weise auch die Tabelle *T* sequenziell gelesen.

Bei der Archivierung mit dem XML-Archivierungs-Operator wird die light-Variante des Bulkdelete-Algorithmus in Verbindung mit den temporären Tabellen eingesetzt, um die Daten in den produktiven Tabellen zu löschen. Dabei werden delete-Ausdrücke der Form

```
delete from <produktive Tabelle T> where <T.Schlüssel> in
      (select <T.Schlüssel>
       from   <produktive Tabelle T>, <temporärer Tabelle D>
       where  <T.Schlüssel>=<D.Schlüssel>
       order by <T.RID>)
```

verwendet. Die light-Variante ist im Idealfall[3] genauso effizient wie der Bulkdelete-Operator und ist auch in allen übrigen Fällen den traditionellen "tuple-at-a-time"-Ansätzen überlegen. Dies belegen die im folgenden Abschnitt beschriebenen Messungen.

4 Beispiel-Implementierung und Messungen

Wir haben eine Reihe von Tests mit einer Beispiel-Implementierung des XML-Archivierungs-Operators durchgeführt. Für unsere Beispiel-Implementierung haben wir den XML-Archivierungs-Operator als *java stored procedure* realisiert. Als zugrundeliegende Datenbank benutzten wir ein sehr weit verbreitetes, kommerzielles Datenbanksystem. Wir haben 2 Varianten des XML-Archivierungs-Operators implementiert:

- Traditionell (**trad**): Als Löschmethode wurde hier die traditionelle Löschmethode verwendet.
- Bulkdelete *light* (**light**): Als Löschmethode wurde hier die light-Variante des Bulkdelete-Algorithmus verwendet.

Bei den Messungen wurde untersucht, wie hoch der Performancegewinn beim Einsatz der light-Variante des Bulkdelete als Löschmethode bereits ist. Als Messdaten wurden die

[3] Im Idealfall ist nur ein Index vorhanden und es liegt eine Ballung der Daten bzgl. der indizierten Spalte vor.

Daten eines von uns im Rahmen einer Kooperation mit SAP entwickelten Archivierungsbenchmarkes verwendet. Die Daten sind an Finanzbuchhaltungsdaten eines SAP-Systems (R/3 4.6C) angelehnt. Die Datenbank lief auf einer SUN Enterprise 450 mit 2 GB Hauptspeicher und 4 Prozessoren a 400 MHz. Die Datenbank belegte 512 MB Hauptspeicher. Als Speichermedium wurde ein SUN A1000 RAID mit 500 GB Speicherkapazität und RAID Level 5 verwendet.

Um die Auswirkungen der Komplexität des betriebswirtschaftlichen Objektes auf den Archivierungsvorgang zu untersuchen, haben wir die Laufzeiten bei 4 unterschiedlichen Objekten gemessen. Die Objekte unterscheiden sich sowohl in der Anzahl der Tupel pro Tabelle, als auch in der Breite der Tupel, d.h. im Verhältnis Daten zu Null-Werten innerhalb eines Tupels (siehe Tabelle 1).

	Tupel stark gefüllt	Tupel wenig gefüllt
wenige Tupel pro Tabelle	Objekt 1	Objekt 2
viele Tupel pro Tabelle	Objekt 3	Objekt 4

Tabelle 1: Definition der Objekttypen

Bei den ersten Messungen wurde die Anzahl der archivierten Objekte variiert. Die Datenbasis hatte eine Größe von 10.000 Objekten. Die Daten jedes der Objekte waren auf 7 Tabellen verteilt. Die Abbildungen 13 und 14 zeigen die Laufzeiten der Löschoperationen bei der Archivierung der verschiedenen Objekttypen.

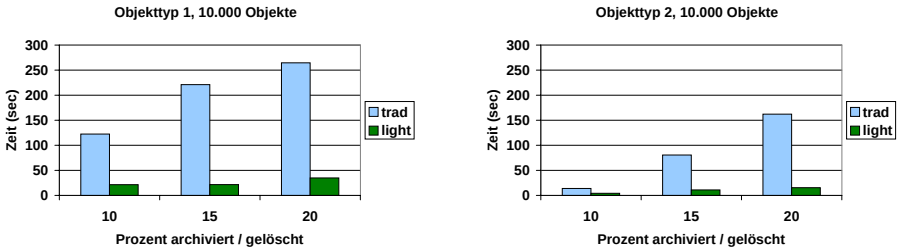


Abbildung 13: Laufzeiten der Löschoperationen für Objekttyp 1 und 2

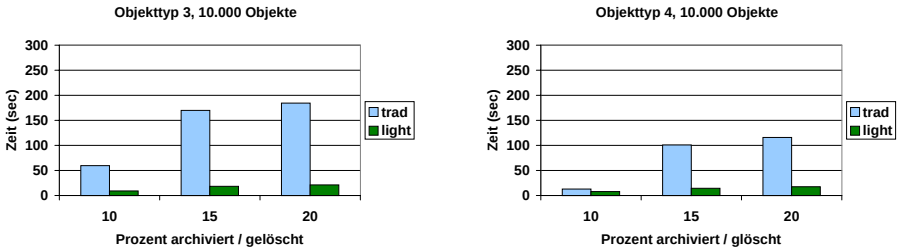


Abbildung 14: Laufzeiten der Löschoperationen für Objekttyp 3 und 4

Wie zu sehen ist, nimmt die Laufzeit der Löschoperation bei der traditionellen Löschmethode (trad) mit zunehmender Menge zu löschender Tupel stark zu. Bei der light-Variante

sind die Laufzeiten wesentlich kleiner, und sie steigen auch mit zunehmender Anzahl gelöschter Einträge weniger stark an. Dies liegt daran, dass die light-Variante jede Hintergrundspeicherseite maximal einmal liest. Die Laufzeit ist deshalb durch die Anzahl der Hintergrundspeicherseiten beschränkt. Bei der traditionellen Variante kann jede Löschung eines Eintrages zu einem Zugriff auf eine andere Hintergrundspeicherseite führen (Random IO). Die Laufzeit ist also hier nur durch die Anzahl der Tupel in der Tabelle beschränkt, die in der Regel sehr viel größer ist als die Anzahl der von der Tabelle belegten Hintergrundspeicherseiten.

In Abbildung 15 sind die Laufzeiten für die XML-Dokument-Generierung zu sehen. Wie zu sehen ist, sind die Zeiten für die XML-Dokument-Generierung wesentlich höher als die Löschzeiten (Minuten gegenüber Sekunden). Obwohl die Zeit für die Datengenerierung dominiert, sind die Löschzeiten keineswegs vernachlässigbar, denn während der Löschphase ist die Tabelle in der Regel komplett mittels einer Schreibsperre gesperrt und daher für die zeitkritischen OLTP-Anwendungen nicht verfügbar. Viele Datenbankensysteme wechseln nämlich bei derart datenaufwendigen Transaktionen, wie einer Massenarchivierung, von einer kleineren Sperr-Granularitäten (einzelner Eintrag, Seite) zur nächst höheren (Seite, komplette Tabelle), um den Aufwand für die Verwaltung der Sperren zu minimieren. Bei der Datengenerierung ist zwar auch die ganze Tabelle gesperrt, aber nur mittels einer Lesesperre. Andere Transaktionen können die Tabellendaten also während der XML-Generierung lesen.

Einige Datenbankhersteller bieten die Möglichkeit an, Datenbankprozesse auf dem rufenden Client zu starten. Dadurch wird der Datenbankrechner entlastet und die Ressourcen des Clients besser ausgenutzt. Wir haben deshalb eine Variante des XML-Archivierungs-Operators implementiert, bei der die speicherintensive XML-Generierung auf dem rufenden Client ausgeführt wird (**client**) und sie mit der serverseitigen Ausführung (**server**) verglichen. Wie in Abbildung 16 zu sehen ist, profitierte die **client**-Variante von der exklusiven Nutzung der Ressourcen des Clients sehr stark.

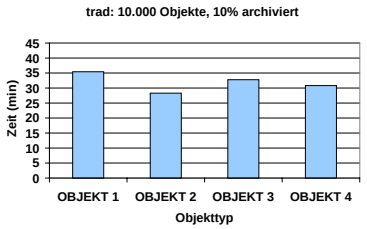


Abbildung 15: Laufzeiten der XML-Dokument-Generierung

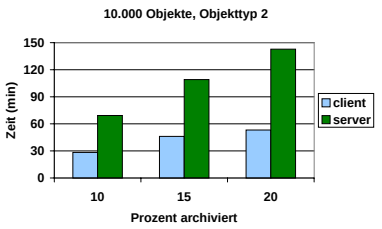


Abbildung 16: Vergleich serverseitige mit clientseitiger XML-Generierung

5 Zusammenfassung

Einer der wichtigsten Einsatzbereiche relationaler Datenbanksysteme liegt in der Verwaltung und Auswertung betriebswirtschaftlicher Daten. Die Datenbanksysteme dienen dabei als Datenspeicher für betriebswirtschaftliche Anwendungssysteme, wie SAP R/3. Anwen-

dungsspezifische Vorgänge, wie z.B. das Definieren und Sperren von betriebswirschaftlichen Objekten, werden von heutigen Datenbanksystemen praktisch nicht unterstützt. Es ist deshalb notwendig, die Schnittstellen der Datenbanken zu erweitern, anzupassen und zu vereinfachen. Ein erster Schritt in diese Richtung sind die XML-Schnittstellen und die objekt-relationalen Erweiterungen, die in fast allen Datenbanksystemen inzwischen zu finden sind.

Wir haben in dieser Arbeit eine Erweiterung relationaler Systeme vorgeschlagen, für die noch keine entsprechende Schnittstelle auf Datenbankebene existiert: den XML-Archivierungs-Operator. Unter Archivierung versteht man dabei das Verschieben der Daten von selten benötigten betriebswirtschaftlichen Objekten aus den produktiven Datenbanksystemen auf kostengünstigere Tertiärspeichersysteme. Dadurch werden die Datenbasis der produktiven Datenbank verkleinert, die Leistung der Datenbank erhöht und letztlich Kosten gesenkt.

Der vorgeschlagene XML-Archivierungs-Operator erlaubt es, im Gegensatz zu den auf Applikations-Ebene implementierten Archivierungskomponenten, den gesamten Archivierungsablauf auf Datenbank-Ebene auszuführen und die Daten als XML-Dokumente abzulegen. Dadurch wird der Datenverkehr zwischen Anwendung und Datenbank verringert. Durch die Ablage als XML wird die Portabilität der Daten erhöht und die Implementierung separater Archivierungskomponenten unnötig gemacht. Vor allem die Ablage als XML macht die Daten nicht nur für andere Anwendungen verfügbar, sondern minimiert auch den Aufwand (und damit die Kosten) bei einem Versionswechsel auf Anwendungsseite erheblich. Der XML-Archivierungs-Operator nutzt intelligente Löschalgorithmen, was den Löschvorgang erheblich beschleunigt und somit den Durchsatz des Gesamtsystems erhöht.

Danksagungen

Wir danken Herrn Klaus Zimmermann, Herrn Thomas Wondrak, Herrn Stefan Krompaß und Herrn Constantin Holzner für die Implementierung des Prototypen und des XML-Archiv-Browsers sowie für die Durchführung der Messungen. Desweiteren danken wir Herrn Wolfgang Becker und Herrn Dr. Ulrich Marquard für die gute Zusammenarbeit im Rahmen des Terabyte-Projektes. Unser Dank gilt außerdem den anonymen Gutachtern für die hilfreichen Kommentare.

Literaturverzeichnis

[ABS99] S. Abiteboul, P. Buneman, and D. Suciu. *Data On The Web, From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1999.

[BEK+00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. `http://www.w3.org/TR/SOAP`, May 2000.

[CFI+00] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Publishing Object-Relational Data as XML. In *Proceedings of the Third International Workshop on the Web and Databases*, pages 105–110, Dallas, USA, May 2000.

[DFF+99] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. In *Proceedings of the Eighth International World-Wide Web Conference*, 1999.

[FTS00] M.F. Fernandez, W.-C. Tan, and D. Suciu. "SilkRoute: Trading between Relations and XML". In *Int'l World Wide Web Conf. (WWW)*, Amsterdam, Netherlands, May 2000.

[GKKZ01] A. Gärtner, A. Kemper, D. Kossmann, and B. Zeller. Efficient Bulk Deletes in Relational Databases. In *Proc. IEEE Conf. on Data Engineering*, pages 183–192, Heidelberg, Germany, 2001.

[Her97] A. Herbst. *Anwendungsorientiertes DB-Archivieren. Neue Konzepte zur Archivierung in Datenbanksystemen*. Springer Verlag, 1997.

[JDO] JDOM Projekt. `www.jdom.org`.

[KN99] K. Küspert and J. Nowitzky. Partitionierung von Datenbanktabellen (Aktuelles Schlagwort). *Informatik Spektrum*, 22(2):146–147, April 1999.

[KS98] K. Küspert and R. Schaarschmidt. Archivierung in Datenbanksystemen (Das aktuelle Schlagwort). *Informatik Spektrum*, 21(5):277–278, October 1998.

[LS98] J. Lufter and R. Schaarschmidt. Auswirkungen von Schemaänderungen einer Datenbank auf die datenbanksystem-integrierte Archivierung. Forschungsergebnisse der Fakultät für Mathematik und Informatik Math/Inf/98/07, Institut für Informatik, Friedrich-Schiller-Universität Jena, March 1998.

[Moh02] C. Mohan. An Efficient Method for Performing Record Deletions and Updates Using Index Scans. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 940–949, HongKong, China, August 2002.

[Now99] J. Nowitzky. Partitionierung relationaler Datenbanktabellen und deren Anwendung für die Datenarchivierung. In *Tagungsband des 11. Workshop „Grundlagen von Datenbanken", Luisenthal/Thüringen, Mai 1999*, Jenaer Schriften zur Mathematik und Informatik, Math/Inf/99/16, pages 77–81, Friedrich-Schiller-Universität Jena, May 1999.

[Now01] J. Nowitzky. Partitionierungstechniken in Datenbanksystemen: Motivation und Überblick. *Informatik Spektrum*, 24(6):345–356, December 2001.

[Rys01] M. Rys. State-of-the-art XML Support in RDBMS: Microsoft SQL Server's XML Features. In *IEEE Data Engineering Bulletin, Vol 24, No 2*, pages 3–11. June 2001.

[SBB+02] H. Stefani, B. Brinkmöller, G. Buchmüller, G. Fischer, M. Fischer, R. Gentinetta, A. Herbst, J. Nolte-Bömelburg, T. Pferdekämper, G. Scherer, and P. Zimmerer. *Datenarchivierung mit SAP*. SAP PRESS, 2002.

[SBH+98] R. Schaarschmidt, K. Bühnert, A. Herbst, K. Küspert, and R. Schindler. Konzepte und Implementierungsaspekte anwendungsorientierten Archivierens in Datenbanksystemen. *Informatik Forschung und Entwicklung*, 13(2):79–89, June 1998.

[Sch01] R. Schaarschmidt. *Archivierung in Datenbanksystemen: Konzept und Sprache*. Teubner-Reihe Wirtschaftsinformatik. Verlag B. G. Teubner, Stuttgart, Leipzig, Wiesbaden, 2001.

[SR97] R. Schaarschmidt and W. Röder. Datenbankbasiertes Archivieren im SAP System R/3. *Wirtschaftsinformatik*, 39(5):469–477, October 1997.

[SSB+01] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as XML documents. *The VLDB Journal*, 10(2-3):133–154, 2001.

[VW02] V. Votsch and M. Walter. Oracle XML DB, 2002. www.oracle.com/ip/deploy/database/oracle9i/collateral/xmldb_buswp.pdf.

[XML00] XML Schema, April 2000. `http://www.w3.org/xml/Schema`.

[ZK02] B. Zeller and A. Kemper. Experience Report: Exploiting Advanced Database Optimization Features for Large-Scale SAP R/3 Installations. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, HongKong, China, August 2002.

A Logical, Transparent Model for Querying Linked XML Documents

Wolfgang May
Institut für Informatik*
Universität Göttingen, Germany
may@informatik.uni-goettingen.de

Dimitrio Malheiro
Institut für Informatik
Universität Freiburg, Germany
malheiro@informatik.uni-freiburg.de

Abstract: The W3C XML Linking Language (XLink) provides a powerful means for interlinking XML documents all over the world. While the effects when browsing through linked XML documents are well-defined, there is not yet any proposal how to handle interlinked XML documents that make use of the XLink language from the database point of view, i.e., considering the data model and navigation/querying aspects. From the database (and in general, querying) point of view, elements with linking semantics can be seen as *virtual* XML subtrees, i.e., *XML views*. Compared with classical databases, i.e., SQL and relational data, the situation of having links *inside* the data is new. We define a *logical, transparent* data model for linked documents. Queries are then formulated in standard XPath against the logical model. We propose additional attributes using the dbxlink (database-xlink) namespace for specifying the mapping from XLinks to the logical model.

1 Introduction

XML data instances are structured as trees, consisting of elements and attributes. The data is *self-describing*, i.e., each data item consists of a name and data contents (cf. the excerpt of the MONDIAL XML database [May01b] given in Figure 1 that will be used for illustrations throughout the paper).

XPath [XPa01] is the common language for addressing node sets in XML documents; we assume that the reader is familiar with XPath. It provides the base for several languages in the XML world, e.g., the query language XQuery, and for XLink. The core XML/XPath concept already provides unidirectional intra-document references by `ID/IDREF` attributes.

Example 1 *Consider the query "search all names (abbreviations) of organizations such that the headquarter city of the organization is also the capital of one of its member countries".*

The query is expressed in XPath as

```
//organization[@headq⇒city = members/@country⇒country/@capital⇒city]/@abbrev .
```

XML data is not required to be self-contained on an individual server, but may include *links* to XML data on other servers. With XLink, the targets of the links are given in XPath-like syntax within the XML data. When querying such distributed XML data, the query language must support following the links, and it must be clear what the *logical* schema of the accessible data as a whole is. This aspect, i.e., extracting a query expression from the intermediate answer that must be evaluated to continue the query evaluation, did not occur before in databases: SQL databases do not contain queries in their data fields.

*On leave from Universität Freiburg.

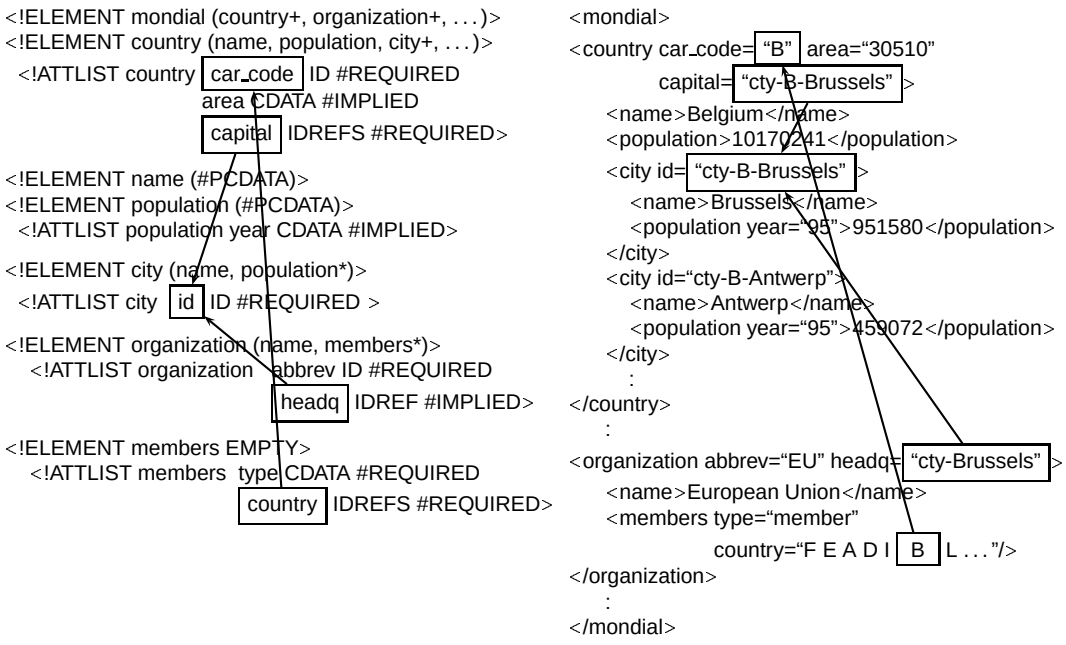


Figure 1: Excerpt of the MONDIAL XML database [May01b]

2 Linked XML Documents

XPointer and XLink specify how to *express* inter-document links in XML. XPointer [XPt00] is a specialized extension of XPath for selecting parts of XML documents – which are not necessarily sets of nodes. The XPointer concept combines the URL document addressing mechanism with an extension of the XPath mechanism for addressing fragments of the document. XPointer "hyperlink" addresses are of the form *url#ext-xpath-expr* . For this work, we restrict ourselves to standard XPath expressions as pointers, i.e., our XPointers are of the form *url#xpath-expr*. E.g., the following XPointer addresses the *country* element that has a *car_code* attribute with value "B" in the document with the url `www.ourserver.de/Mondial/mondial.xml`:

www.ourserver.de/Mondial/mondial.xml#descendant::country[@car_code="B"]

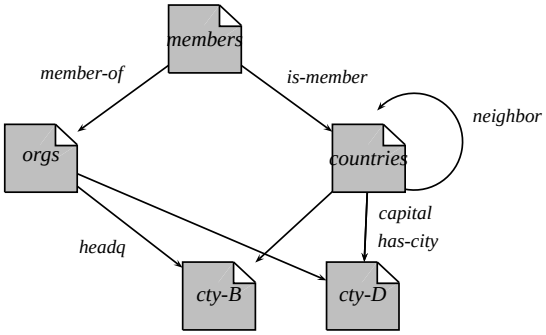
The XML linking semantics is specified in the *XML Linking Language (XLink)* [XLi00] by providing special tags in the xlink: namespace that tell an application that an element is equipped with link semantics. Arbitrary elements can be declared to have link semantics by equipping them with an xlink:type attribute and suitable additional attributes and subelements from the xlink: namespace. The xlink:type attribute selects between basic types of links: *simple links* extend the semantics known from <A href="...">. Their xlink:href attribute selects a target of the individual link instance, allowing for addressing nodes inside the target document by an XPointer. *Extended links* allow for grouping of targets, and also for specifying relationships *between* such targets. So far, XLink provides just a *syntactic representation* of references.

The additional xlink: attributes xlink:actuate and xlink:show specify the *behavior* of a link, i.e., its activating event and the triggered action. This behavior is tailored to the use of links when *browsing*, it does not cover the requirements of querying XML instances.

XLink does not provide any information about the *data model* or how queries are stated: there is not yet an official proposal (i) how to add link semantics to the actual data model, e.g., the DOM or the XML Query Data Model [XMQ01a], and (ii) how to *handle* links in queries and applications (which in part depends on the data model, but orthogonally, evaluation strategies have to be defined). In this paper, we focus on the *data modeling* aspect – which is then the base for formulating queries. We investigated the evaluation aspects of distributed queries in [May02b].

Example 2 *In the following, we illustrate the use of the different types of links by a "distributed" version of* MONDIAL *where all countries, all cities of a country, all organizations, and all memberships are stored in separate files.*

- *countries.xml (all countries)*
- *cities-car-code.xml (the cities for each country)*
- *organizations.xml (all organizations)*
- *memberships.xml (relates countries and organizations)*



Example 3 (Cities) *The cities-country.xml documents are very simple. Note that cities even do not have an ID; we assume that their name inside a country is unique. Below, the DTD and an excerpt of cities-B.xml is given:*

```
<!ELEMENT cities (city+)>
<!ELEMENT city (name, population*)>

<!ELEMENT name (#PCDATA)>
<!ELEMENT population (#PCDATA)>
 <!ATTLIST population year CDATA #IMPLIED>
```

```
<cities>
 <city> <name>Brussels</name>
    <population year="95">951580</population>
 </city>
 <city> <name>Antwerp</name>
    <population year="95">459072</population>
 </city>
    :
</cities>
```

Simple Links. A simple link is similar to the HTML <A href="..."> construct. It contains only a single pointer, but note that this pointer can address one or more elements.

Example 4 (Countries and Cities) *The country data is stored in countries.xml. A country has a capital and several cities. The capital is referenced by a simple link. The cities are also referenced by a simple link that addresses a set of nodes.*

```
<!ELEMENT countries (country+)>
<!ELEMENT country (. . . , capital, cities, . . . )>
```

```
  <!ATTLIST country  car_code ID #REQUIRED>
<!ELEMENT capital EMPTY>
  <!ATTLIST capital xlink:type (simple|extended|locator|arc)  #FIXED "simple"
                    xlink:href  CDATA #REQUIRED>
<!ELEMENT cities EMPTY>
  <!ATTLIST cities xlink:type (simple|extended|locator|arc)  #FIXED "simple"
                   xlink:href  CDATA #REQUIRED>

<countries>
  <country car_code="B"> <name>Belgium</name>
    <capital href="file:cities-B.xml#//city[name='Brussels']"/>
    <cities href="file:cities-B.xml#//city"/>
    :
  </country>
    :
</countries>
```

Example 5 (Headquarters of Organizations) *The file organizations.xml in the distributed version does not contain information about memberships. Thus, only the @headq attribute of organizations is replaced by a headq subelement which is a simple link:*

```
<!ELEMENT organizations (organization+)>
<!ELEMENT organization (name, headq)>
  <!ATTLIST organization  abbrev ID #REQUIRED>
<!ELEMENT headq EMPTY>
  <!ATTLIST headq xlink:type (simple|extended|locator|arc)  #FIXED "simple"
                  xlink:href  CDATA #REQUIRED>

<organizations>
  <organization abbrev="EU"> <name>European Union</name>
    <headq xlink:href="file:cities-B.xml#//city[name='Brussels']"/>
  </organization>
    :
</organizations>
```

Additionally, there are *inline extended links*, and *out-of-line* extended links. The latter allow to create references not only inside documents, but also to create XML instances that consist *only* of links between other documents (e.g., memberships of countries in organizations).

3 Querying along Links

Each link can be seen as a view definition – possibly recursively containing further links; in this case, the view may be even infinite (due to cyclic references). Whereas in SQL, a view or a database link appears as a table or a database schema that easily fits with the language syntax and semantics, links as *tree view definitions* embedded into the data itself need some special handling.

We propose a *logical* data model where the link elements are regarded to be *transparent*:

the linked XML sources are mapped to a logical model that consists of a single XML tree. This logical model can then be processed with standard XPath, XQuery, or XSLT. This *logical* data model silently replaces link elements of the types xlink:simple and xlink:locator by the result sets of their XPointers, and elements of the types xlink:extended and xlink:arc are assigned with a (re)structuring semantics. Thus, the logical, transparent model is already a kind of a view of the data. The view is generated from the input documents only by restructuring the tree at the XLink elements. Thus, it does not require any separate query. Figure 2 illustrates the general intuition of replacing references by tree views.

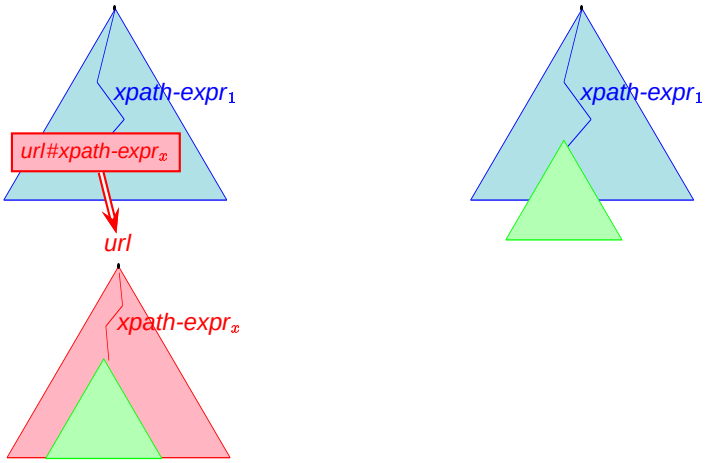


Figure 2: Extended XML Data Model with XLink Elements

The *external*, *logical* schema is induced in a well-defined way (that is described in more detail below) by the structure of the entry document, and by the structures of the linked documents. This *external schema* in turn induces the possible queries against the entry data source. Evaluation of these queries maps them back to the underlying documents, evaluating parts of the queries against the linked sources.

Example 6 (Motivation) *A simple, generic transparent model is obtained by always* replacing *the XLink elements by the XML contents that is referenced by them (details are described in the rest of the paper). In such a model, the sample query reads as*

```
(**)   document("memberships.xml")//membership
           [organization/headq = country/capital]/organization/@abbrev
```

The mapping between distributed, linked XML data, and a single XML instance finds applications in both directions:

- mapping a set of distributed, linked documents into a single, *logical* XML instance that is then queried in XPath, and
- a given XML instance can be distributed over several instances that are connected by

XLinks *without* changing its *logical model* – i.e., all queries yield the same answer against the original instance as against the logical model of the distributed database.

There is no *generic* intuitive transparent model. We propose a language extension to XLink that uses attributes – in the same way as e.g. xlink:show – to specify for each link element how it should be mapped to a transparent model.

4 Transparent Links: Modeling Switches

Depending on the link type, there are alternatives for mapping it to the logical model:

Simple Links. For simple links, the link consists of the link element – providing a name, and possibly attributes and element contents – and an XPointer:

- the *XLink element itself* can either be (i) kept, or (ii) dropped, or (iii) dropped and its attributes are replicated into the (resulting) subelements, or (iv) transformed into a (reference) attribute (its attributes are replicated into the (resulting) subelements).
- the result of *evaluating* an XLink element or a reference can be inserted as an element, or for each element of the result, its *attributes* and *contents* can be inserted an existing "element hull" that is provided by the surrounding XLink element.

Inline Extended Links. For inline extended links, "the link" consists of a grouping structure and a sequence of locators. Both can be handled separately in the above way.

Out-of-Line Links and Arcs. An extended, out-of-line link element is a collection of (i) locators and (ii) arcs. The surrounding link itself can be kept or dropped, and either

- the locator elements themselves can be ignored in the transparent model and the xlink:from and xlink:to attributes of the arcs are materialized as subelements, or
- the locators are kept (and handled with the same alternatives as simple links) and the arc is translated into reference attributes to the locators.
- It is possible to introduce new names for the xlink:from and xlink:to roles of arcs.

4.1 Specification in the dbxlink Namespace

We use the dbxlink:transparent attribute (denoting the database aspect of XLink) for specifying how the respective link elements are treated in the logical model (a formal characterization and further examples can be found in [May02a]):

For all XLink elements:

- keep-element: the XLink element itself is kept (without the XLink attributes), and contents and/or attributes are inserted.
- drop-element: the XLink element is dropped, i.e., replaced by the results of evaluating its attributes and contents.
- keep-attributes: the XLink element itself is dropped, the non-XLink attributes are kept

and added to each referenced element.

For simple links and locators (i.e., those elements that have an href attribute):

- insert-elements inserts the whole referenced element(s),
- insert-contents inserts the contents and attributes of the referenced element(s) into the surrounding element,
- insert-nothing does nothing: when the locator is actually only used for arcs, it should not be considered itself in the transparent model.
- make-attribute: the XLink element itself is dropped, instead a reference attribute is added to the surrounding element that yields the referenced elements (which are added to the logical instance "somewhere"). The non-XLink attributes of the link element are added to the referenced element(s).

Each arc contains a specification how to handle the from-locator and the to-locator: the dbxlink:transparent attribute can contain the values

- keep-from, drop-from, keep-from-attributes or make-from-attribute,
- keep-to, drop-to, keep-to-attributes or make-to-attribute,
- from-elements or from-contents, to-elements or to-contents,
- optionally, dbxlink:from-role and dbxlink:to-role specify what names are used for the from and to references.

Default Values. In general, data sources are given on the Web without dbxlink attributes. For that case, we propose the following default setting for dbxlink:transparent that keeps the names and possible additional attributes of the navigation elements and fills them with the contents of the referenced elements..

- keep-element for simple links, extended links, and arcs,
- insert-contents for simple links,
- keep-from/keep-to and from-contents/to-contents for arcs,
- drop-element and insert-elements for locators in inline extended links,
- drop-element and insert-nothing for locators in out-of-line extended links since they are just auxiliary.

4.2 Examples

Single-Target Simple Links. If a single target element is linked, it should often either appear instead of the link, or its contents should be integrated into the link element.

Example 7 (Headquarters of Organizations) *Consider again Example 5.*

The default settings for dbxlink:transparent (i.e., "keep-element insert-contents") map the contents and attributes of the referenced city element into the headq link element. The corresponding excerpt of the logical transparent instance of organizations.xml looks as follows, using the contents of the city element that represents Brussels in cities-B.xml:

```
<organizations>
  <organization abbrev="EU"> <name>European Union</name>
    <headq id="cty-B-Brussels">
      <name>Brussels</name>
      <population year="95">951580</population>
    </headq>
  </organization>
    :
</organizations>
```

Then the name of the headquarter city of an organization can be selected by

document("organizations.xml")/organization[@abbrev="EU"]/headq/name .

Example 8 (Headquarters of Organizations – Alternative Modeling) *Consider again Examples 5 and 7. Another modeling alternative is to resolve the headq link element into a* reference attribute*. Then, the source document looks as follows, containing the dbxlink:transparent specification:*

```
<organizations>
  <organization abbrev="EU"> <name>European Union</name>
    <headq dbxlink:transparent="make-attribute"
           href="file:cities-B.xml#//city[name='Brussels']"/>
  </organization>
    :
</organizations>
```

The logical instance then looks as follows (referenced elements are added "somewhere"):

```
<organizations>
  <organization abbrev="EU" headq="localcopyofbrussels01">
    <name>European Union</name>
  </organization>
    :
  <!-- here, the "imported" elements are stored -->
    <city id="localcopyofbrussels01"> <name>Brussels</name> ... </city>
    :
</organizations>
```

Then, the query can be stated in the same way as for the non-distributed document (Fig. 1):

document("organizations.xml")/organization[@abbrev="EU"]/@headq⇒city/name

Example 9 *Similar considerations as above for the headquarter cities of organizations hold for the capital reference in Example 4: the capital reference can either be mapped to a subelement or to an attribute. Then, the queries read as*

document("countries.xml")/country[@car_code="B"]/capital/name *or*
document("countries.xml")/country[@car_code="B"]/@capital⇒city/name *respectively*.

Further examples -including arcs- can be found in [May02a]).

5 Conclusion

We have discussed a *logical model* for linked XML documents that makes the links transparent. Queries are then stated against the logical schema without bothering the user about the distributed nature of the data, and how to handle the links. The intended logical schema can be specified by the owner of the XML documents by appropriately setting the attributes for the XLink elements in the dbxlink namespace. Mainly, we see two scenarios where the logical model is relevant:

- Querying: Given autonomous sources that are interlinked by XLink, the above defines a logical model for stating querying against such sources (in general, when no dbxlink attributes are given, the defaults specify the modeling).
- Database redesign: Given a single XML instance (according to a given, public schema) that should be split/distributed over several instances, the references between its parts are usually expressed by XLink (cf. Ex. 2). The attributes in the dbxlink namespace can then be used for retaining the original *logical model* and *external schema* wrt. the user – i.e., all queries can remain unchanged.

Recall that the transparent model is just a *logical, virtual* model. The decision whether it is materialized depends on the evaluation strategies (see [May02b]).

Materialization. For validating the above definition of the transparent model, an XSLT script [Mal02] has been created according to the recursive definition for the generation of the logical instance. Starting with a given XML instance, it processes the XML tree recursively. For each element that is equipped with XLink functionality (i.e., an xlink:type attribute), an appropriate template is applied that transforms the link element. In case of simple links and locators, the xlink:href attribute is evaluated, and the result list is processed recursively and the result is included into the result document. For arcs, the from and to attributes are evaluated, the corresponding locators are processed, and the results are again included into the result. In case of infinite, cyclic logical instances, the script returns a warning when a link/element pair is processed that is already on the current path and stops the recursion.

Note that materializing the transparent instance is only an intermediate step to define a *logical* instance (with a logical schema) as a base for querying.

Query Evaluation. Based on the promising results, an extension of the LOPiX system [May01a] is under work for investigating the handling of links in an XPath-based environment. An implementation in a standard XML database is planned. In these cases, the logical instance will not be materialized, but the evaluation of the query is split at the links, subqueries/views are evaluated wrt. the referenced documents, and the answers are then recombined. There are several possibilities, concerning *when* and *where* the views defined by links are evaluated, and what results may be cached. These issues are discussed in [May02a, May02b], also proposing the use of additional attributes in the dbxlink namespace for specifying evaluation strategies.

Related Work. The XML query languages XML-QL [DFF+99] and XQuery [XQu01] (and related approaches) allow to express "distributed" queries (e.g., for information inte-

gration) as *joins* of several queries to different sources. But, although the *W3C XML Query Requirements* [XMQ01b] explicitly state that *"3.4.12: Queries MUST be able to traverse intra- and inter-document references"*, neither XPath nor XQuery (and also not the earlier XML-QL) support navigation along XLink references. There is not yet any other work on querying linked XML instances.

XML Linking and Style [Wal01] is concerned with styling linked XML documents via XSL stylesheets. It proposes to add attributes to the xsl namespace that define – refining the behavior specified by the xlink:show attribute – how the *styled result* is embedded into the presentation of the current document. The main difference between *XML Linking and Style* and our approach is that the former operates on the representational, browsing level, where our approach operates on the data model level, defining a "database instance" that can then be queried.

Theoretical aspects of distributed query evaluation for semistructured data are discussed in [Suc02]. The paper does not consider the details how to resolve the links into a logical instance and schema, but focuses on the algorithms and distributed evaluation techniques for queries that use such distributed data.

Bibliography

[DFF+99] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. XML-QL: A Query Language for XML. In *8th. WWW Conference*. W3C, 1999. World Wide Web Consortium Technical Report, `www.w3.org/TR/NOTE-xml-ql`.

[Mal02] Dimitrio Malheiro. Generating a Transparent Instance from Linked XML Documents (XSLT script). Available at `www.informatik.uni-freiburg.de/~may/LinXIS/`.

[May01a] Wolfgang May. LoPiX: A System for XML Data Integration and Manipulation. In *Intl. Conf. on Very Large Data Bases (VLDB), Demonstration Track*, 2001.

[May01b] Wolfgang May. The MONDIAL Database, 2001. `http://www.informatik.uni-freiburg.de/~may/Mondial/`.

[May02a] Wolfgang May. Considerations on Linked XML Document Networks in the Web. Available at `www.informatik.uni-freiburg.de/~may/LinXIS/`, 2002.

[May02b] Wolfgang May. Querying Linked XML Document Networks in the Web. In *11th. WWW Conf.*, 2002. Available at `http://www2002.org/CDROM/alternate/166/`.

[Suc02] Dan Suciu. Distributed Query Evaluation on Semistructured Data. *ACM Transactions on Database Systems (TODS)*, 27(1):1–62, 2002.

[Wal01] N. Walsh (ed.). XML Linking and Style. W3C Note `http://www.w3.org/TR/xml-link-style`, 2001.

[XLi00] XML Linking Language (XLink). `http://www.w3.org/TR/xlink`, 2000.

[XMQ01a] XML Query Data Model. `http://www.w3.org/TR/query-datamodel`, 2001.

[XMQ01b] XML Query Requirements. `http://www.w3.org/TR/xmlquery-req`, 2001.

[XPa01] XML Path Language (XPath) Version 1.0: 1999; version 2.0: 2001. `http://www.w3.org/TR/xpath20`, 2001.

[XPt00] XML Pointer Language (XPointer). `http://www.w3.org/TR/xptr`, 2000.

[XQu01] XQuery: A Query Language for XML. `http://www.w3.org/TR/xquery`, 2001.

T-XPath: Ein zeitliches Modell für XML-Datenbanken

Markus Kalb[1], Kerstin Schneider[2], Günther Specht[1]

[1]Universität Ulm
Abt. Datenbanken und Informationssysteme
{kalb,specht}@informatik.uni-ulm.de

[2]European Media Laboratory GmbH (EML), Heidelberg
Kerstin.Schneider@eml.villa-bosch.de

Abstract: Bisherige XML-Datenbanken erlauben lediglich eine eingeschränkte Unterstützung von zeitlichen Daten und Anfragen. Das hier vorgestellte zeitliche Modell T-XPath erweitert das bisherige Datenmodell und die Anfragesprache XPath um eine flexible und effiziente Modellierung, Verwaltung und Abfragemöglichkeit von zeitlichen Informationen. Als Grundlage dienen abstrakte zeitliche Datentypen (ADT), die die gesamte zeitliche Entwicklungsgeschichte eines Wertes kapseln und zusätzlich auch unscharfe und ungenaue Zeitangaben berücksichtigen können. Die Anfragesprache von T-XPath ist voll abwärtskompatibel zu XPath und stellt für zeitliche Anfragen eine Reihe neuer Operationen und Funktionen zur Verfügung.

1 Einleitung

In vielen Anwendungsgebieten von XML-Datenbanken kommt zunehmend die Anforderung hinzu, auch komplexe zeitliche Daten und Anfragen zu unterstützen. Die bisherigen Möglichkeiten von XML, insbesondere XPath und XSchema, bieten hierfür jedoch lediglich eine sehr eingeschränkte Unterstützung.

Die Verwaltung und Verarbeitung von komplexer zeitlicher Information, welche zum Teil sogar unscharf und ungenau ist, wird beispielweise im Projekt GEIST[1] erforderlich [Kr01], das am European Media Laboratory GmbH (EML) in Kooperation mit dem Fraunhofer Institut für Graphische Datenverarbeitung sowie dem Zentrum für Graphische Datenverarbeitung in Darmstadt durchgeführt wird. Es hat die Entwicklung eines mobilen Augmented-Reality-Systems zum Ziel, welches es erlaubt, Geschichte an Ort und Stelle zu erleben und zu erfahren. Als Beispielszenario wurde der 30jährige Krieg (in Heidelberg) gewählt. Neben dem Erleben von interaktiven Erzählungen auf der Grundlage historischer Information, soll es den Benutzern möglich sein, jederzeit weitergehende historische Information zu erfragen.

Für XML-Datenbanken wurde vom W3C die Anfragesprachen XPath und (darauf aufbauend) XQuery vorgeschlagen [W3C02]. Im folgenden wird die Anfragesprache T-XPath vorgestellt, die XPath um eine flexible und effiziente zeitliche Verwaltung erweitert, die unscharfe und ungenaue zeitliche Information berücksichtigen kann.

Das Papier ist wie folgt gegliedert: In Abschnitt 2 werden die bisherigen Möglichkeiten von XPath zur Unterstützung von zeitlichen Informationen dargestellt. In Abschnitt 3 wer-

1. Das GEIST Projekt wird gefördert vom BMBF (01 IRA 12A, 01 IRA 12B, 01 IRA 12C) und von der Klaus Tschira Stiftung (KTS).

den neue, flexible zeitliche Datentypen vorgestellt, die zusammen mit dem in Abschnitt 4 definierten temporalen Datenmodell die Grundlage von T-XPath bilden. In Abschnitt 5 wird die Anfragesprache von T-XPath vorgestellt. Abschließend werden in Abschnitt 6 verwandte Arbeiten betrachtet.

2 Unterstützung von zeitlichen Information in XPath

XPath ist eine Anfragesprache zur Navigation innerhalb der hierarchischen Struktur eines XML-Dokumentes, zur Selektion von Teilen des Dokumentes sowie zur Manipulation der selektierten Daten [W3C02]. Im Folgenden wird die derzeit noch in Arbeit befindliche Version 2.0 zugrunde gelegt.
In XPath werden Zeitangaben durch die Verwendung der einfachen zeitlichen Datentypen aus XSchema definiert: *duration*, *dateTime*, *time*, *date*, *gYearMonth*, *gYear*, *gMonthDay*, *gMonth*, *gDay*. Diese bieten lediglich eingeschränkte Möglichkeiten zur Behandlung von zeitlichen Angaben. So sind die einzelnen Datentypen entsprechend dem gregorianischen Kalender definiert. Jeder Datentyp entspricht einer anderen Granularität des Kalenders. Das hat vielfältige Nachteile: Vergleiche zwischen den unterschiedlichen Datentypen werden dadurch erschwert, Änderungen des Datenformates oder des Kalenders gestalten sich aufwendig, beliebige zeitliche Perioden, wie beispielweise "03.10.01 - 06.10.01", müssen explizit modelliert werden. Des weiteren werden Aggregationen (Coalescing) von zeitlichen Perioden nicht direkt unterstützt. Schließlich sind ungenaue und unscharfe Zeitangaben nicht definierbar. Die bisherigen Möglichkeiten von XPath und XSchema sind für eine robuste und flexible Behandlung von Zeitangaben nicht ausreichend bzw. für eine zeitliche Verwaltung von Information nicht vorhanden. Neue zeitliche Datentypen sowie ein Konzept ihrer Interaktionen mit konkreten Werten werden benötigt um komplexe zeitliche Informationen effizient zu verwalten.

3 Neue zeitliche Datentypen für Zeitangaben in T-XPath

Die in der letzten Dekade im Bereich temporale Datenbanken eingeführten Konzepte bilden die Grundlage für die nachfolgenden Definitionen der zeitlichen Datentypen in T-XPath. Dabei besitzen alle Typen die folgenden grundlegenden Eigenschaften:
Einen zeitlichen Wertebereich: Die Zeit wird als eine eindimensionale, lineare und geordnete Linie betrachtet. Diese ist in diskrete Einheiten, Chronons, unterteilt und ist isomorph zu den ganzen Zahlen $\mathbb{Z}$.
Kalenderunabhängigkeit: Allen Zeitprimitiven liegen Chronons zugrunde. Sie sind somit unabhängig von einem spezifischen Kalender. Umrechnungen in konkrete Kalendarien erfolgen über Abbildungsfunktionen.
Unschärfen: Zeitliche Information ist charakterisiert durch Ungenauigkeit und Unbestimmtheit, da (1.) die Messung und Speicherung der Zeit in Kalendern mit unterschiedlichen Basisgranularitäten zu Ungenauigkeiten führt und (2.) die Information über zeitliche Angaben oft nicht exakt vorhanden ist. Beispielsweise sind geschichtliche Ereignisse häufig ungenau überliefert: "Der Schlossbau begann zwischen 1500 und 1502". Beide Formen der Unschärfe werden durch die neuen Datentypen unterstützt.
Spezielle Variablen ermöglichen die besondere Behandlung bestimmter Zeitangaben.

Beispielsweise bezeichnet die Variable *now* die aktuelle Zeit. Die Variablen *until change* und *since beginning* beschreiben den Anfang und das Ende der Zeitlinie.

3.1 Atomare zeitliche Datentypen

Als Grundlage führen wir folgende drei zeitliche Datentypen in T-XPath ein: Zeitdauer (*interval*), Zeitpunkt (*instant*) und Zeitperiode (*period*).

Zeitdauer (*interval*)

Der Typ *interval* charakterisiert die Dauer eines Ereignisses, z.B. "30 Tage". Eine ungenaue Zeitdauer entspricht z.B. der Aussage "das Ereignis dauerte zwischen 5 und 7 Stunden". Es werden das Minimum d_{min} bzw. Maximum d_{max} von möglichen Werten definiert. Die Zeitangabe ist genau dann exakt, wenn $d_{min} = d_{max}$.

$$interval := \{(d_{min}, d_{max}) \mid d_{min}, d_{max} \in \mathbb{Z}, 0 \leq d_{min} \leq d_{max}\} \quad \textit{(Definition 1)}$$

Zeitpunkt (instant)

Ein Zeitpunkt ist eine Zeitangabe, die der Granularität genau eines Chronons entspricht, z.B. der "01.10.2000" wenn die Granularität des Kalenders einem Tag entspricht. Ungenaue Zeitangaben sind z.B. "Ein Tag in der ersten Woche im Oktober 2000". Ein ungenauer Zeitpunkt wird definiert durch die Angabe des frühest möglichen Zeitpunkts *c* des Ereignisses sowie die Anzahl der möglichen nachfolgenden Zeitpunkte d_Δ.

$$instant := \{(c, d_\Delta) \mid c, d_\Delta \in \mathbb{Z}, d_\Delta \geq 0\} \quad \textit{(Definition 2)}$$

Zeitperiode (period)

Eine Zeitperiode entspricht einer Zeitangabe, die einen Bereich beschreibt während dem ein Ereignis stattgefunden hat, z.B. "Im gesamten März 2001" oder "01.03.01 - 15.03.01". Eine Zeitperiode wird durch die Angabe ihres Anfangs- und Endzeitpunktes definiert. Ungenaue Zeitperioden besitzen einen ungenauen Anfangs und/oder Endzeitpunkt. Zusätzlich gilt die Bedingung, dass der Endpunkt nicht vor dem Anfangszeitpunkt liegen darf. Zeitangaben wie z.B. "Es begann zwischen 1500-1502 und endete zwischen 1600 und 1603" sind damit beschreibbar.

$$period := \{(i_1, i_2) \mid i_1, i_2 \in instant, (before(i_1, i_2) \vee overlaps(i_1, i_2) \vee equal(i_1, i_2))\}^{1} \quad \textit{(Definition 3)}$$

Diese drei neuen atomaren Datentypen ersetzten die bisherigen zeitlichen Datentypen von XSchema respektive XPath.

3.2 Operationen über den atomaren zeitlichen Datentypen

In der Literatur wird eine Vielzahl von zeitlichen Operationen auf zeitlichen Datentypen beschrieben [Al84, Je00, Sn00]. Die Einbeziehung unscharfer Zeitangaben erfordert jedoch eine Erweiterung einiger Operationen. Insbesondere boolesche Operationen müssen um einen dreiwertigen booleschen Datentyp (*3-bool* [CP01]) ergänzt werden. In Tabelle 3.1 wird am Beispiel von zwei Operationen deren Signatur und Semantik defi-

1. Die verwendeten Operationen entsprechen den Operation von J.F. Allen und werden später noch näher erklärt.

niert. Dies sind die Vergleichsoperation (<) für Zeitintervalle und die Operation *before()* aus Allen's Operationen für Zeitperioden [Al84].

Operation	Signatur	Semantik
<	$interval \times interval \rightarrow 3\text{-}bool$	$I_1 < I_2 \equiv \begin{cases} true : I_1.d_{max} < I_2.d_{min} \\ false : I_1.d_{max} > I_2.d_{min} \\ maybe : else \end{cases}$
before	$period \times period \rightarrow 3\text{-}bool$	$before(P_1,P_2) \equiv \begin{cases} true : ((P_1.I_2.c + P_1.I_2.d_\Delta) < P_2.I_1.c) = true \\ false : (P_1.I_2.c \geq (P_2.I_1.c + P_2.I_1.d_\Delta)) = true \\ maybe : else \end{cases}$

Tabelle 3.1: Beispiele für zeitliche Operationen

In T-XPath stehen folgende zeitliche Operationen zur Verfügung:
- arithmetische Operationen auf *interval*, *instant*, *period*
- Vergleichsoperationen auf *interval* (<, =, >,...)
- Allen's Operationen auf *period*, *instant* (*before*, *meets*, *after*,...)
- Cast Operationen (*duration_of_period*)

4 Das T-XPath Datenmodell

4.1 Klassifikation der Objekte des bisherigen XPath-Modells

XPath operiert auf der abstrakten, logischen Struktur eines XML-Dokuments und repräsentiert diese in seinem Modell als eine geordnete Baumstruktur. Innerhalb des Baumes werden Knoten, atomare Werte und Sequenzen als Grundelemente unterschieden. Für die Definition der Semantik der zeitlichen Verwaltung ist eine genauere Klassifikation dieser Objekte erforderlich.
Es gibt sieben Knotenarten: Dokument, Element, Attribut, Text, Namensraum, Prozess-Instruktion und Kommentar. Die Dokument- und Element-Knoten können als einzige weitere Knoten, so genannte Kindknoten, beinhalten und repräsentieren die hierarchische Struktur eines XML-Dokuments. Konkrete Informationen können lediglich in den anderen fünf Knotenarten gespeichert werden. Somit lassen sich Knoten in die zwei Klassen der "strukturierenden" Objekte und der "quantifizierenden" Objekte disjunkt einteilen.

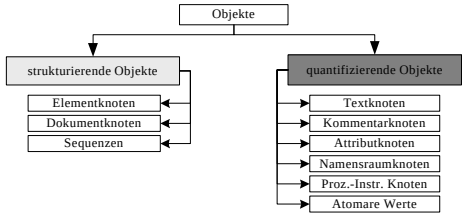


Abbildung 4.1: Vollständige Klassifizierung der XPath-Grundelemente

Die atomaren Werte des Datenmodells von XPath entsprechen atomaren Datentypen und korrespondieren mit den Datentypen aus XSchema. Atomare Werte gehören zu den "quan-

tifizierenden" Objekten. Eine Sequenz entspricht einer geordneten Menge von Knoten oder atomaren Werten, die jeweils auch mehrfach enthalten sein können. Sequenzen sind ‚flach', d.h. sie dürfen keine Sequenzen enthalten, und sind der Klasse der "strukturierenden" Objekte zuzuordnen. Ein vollständiger Überblick über die von uns vorgestellte Klassifikation der Grundelemente findet sich in der Abbildung 4.1.

4.2 Semantik der zeitlichen Verwaltung

Anhand der Klassifikation können zwei unterschiedliche Semantiken der zeitlichen Verwaltung abgeleitet werden:

a) In den strukturierenden Objekten von XPath werden Strukturen zeitlich verwaltet, d.h. es wird die Geschichte von Elementen und somit der Entwicklungsverlauf der Struktur eines XML-Dokuments repräsentiert (Schemaversionisierung).

b) Innerhalb der quantifizierenden Objekte werden konkrete Informationen zeitlich verwaltet. Dies entspricht dem geschichtlichen Verlauf eines konkreten Wertes (Objektversionisierung)

4.3 Die Integration der zeitlichen Verwaltung

Bei der zeitlichen Verwaltung von Information bilden die *Gültigkeitszeit* (*valid time*), wann war ein Wert in der realen Welt gültig, und die *Aufzeichnungszeit* (*transaction time*), wann wurde ein Wert gespeichert, die grundlegenden Zeitarten. Stehen beide zur Verfügung spricht man von *Bitemporaler Zeit*.
Orthogonal dazu werden Tupel- und Attributzeitstempelverfahren zur Verknüpfung der Information mit Zeitangaben unterschieden [Sk97, JE00]. Neuere Ansätze erweitern das Attributzeitstempelverfahren dahingehend, dass ein zeitabhängiger Wert als ein abstrakter zeitlicher Datentyp (ADT) modelliert wird [Er99, Gü00, CR01]. Die zeitliche Verwaltung findet ausschließlich innerhalb des ADT statt und muss nicht, wie bei den bisherigen Verfahren, explizit realisiert werden. Ein ADT erweitert einen nicht-zeitabhängigen Datentyp um eine zeitliche Verwaltung, wobei dessen ursprüngliche Eigenschaften bestehen bleiben und lediglich um neue zeitliche Eigenschaften ergänzt werden. Die gekoppelte Speicherung von konkreten Werten und Zeitangaben in einem Datentyp, ermöglicht effiziente Algorithmen insbesondere für Operationen, die gleichzeitig auf zeitlichen und konkreten Werten operieren (z.B. Änderungsrate eines Wertes). Diese Operationen waren mit den bisherigen Verfahren nur schwer oder nicht effizient lösbar [Er99]. Daher bilden ADTs die Grundlage für das T-XPath Modell.

4.4 Die abstrakten zeitlichen Datentypen

Die zeitliche Verwaltung mit ADTs erweitert lediglich das Typsystem von T-XPath. D.h. es werden keine neuen zeitlichen Elemente eingeführt, sondern neue Datentypen und Operationen, die in den quantifizierenden Objekten verwendet werden können. Somit wird die hierarchische Baumstruktur des zugrundeliegenden XPath-Datenmodells durch die zeitliche Verwaltung nicht beeinflusst.

In T-XPath werden für alle bisherigen Datentypen (z.B. *string*, *integer*, etc.) drei korrespondierende abstrakte zeitliche Datentypen zur Verfügung gestellt, die deren Gültigkeitszeit-, Aufzeichnungszeit- oder bitemporale Entwicklungsgeschichte repräsentieren. Die jeweiligen Eigenschaften der Zeitarten, beispielsweise keine Lücken in der Aufzeichnungszeit zu erlauben, sind für jeden der Datentypen formal definiert. Im folgenden sind am Beispiel des Datentyps *string*, dessen korrespondierende zeitliche Datentypen definiert. Die zusätzlichen Bedingungen in den Definitionen von *t_string* und *vt_string* stellen sicher, dass keine Löcher in der Aufzeichnungszeitgeschichte existieren.

validtime_string *(v_string)* *(Definition 4)*

$$v\_string := \left\{ B = \{(v_1, vt_1), \ldots, (v_m, vt_m)\} \middle| \begin{array}{l} v_i \in string, vt_i \in interval \cup instant \cup period \\ 1 \le i \le m, m \in \mathbb{N} \end{array} \right\}$$

transactiontime_string *(t_string)* *(Definition 5)*

$$t\_string := \left\{ B = \{(v_1, tt_1), \ldots, (v_m, tt_m)\} \middle| \begin{array}{l} v_i \in string, tt_i \in period, 1 \le i \le m, m \in \mathbb{N}, \\ 1 \le k \le m-1, meets(tt_k, tt_{k+1}) = true \end{array} \right\}$$

bitemporal_string *(vt_string)* *(Definition 6)*

$$vt\_string := \left\{ B = \{(v_1, vt_1, tt_1), \ldots, (v_m, vt_m, tt_m)\} \middle| \begin{array}{l} v_i \in string, tt_i \in period, vt_i \in interval \cup instant \cup period \\ 1 \le i \le m, m \in \mathbb{N}, 1 \le k \le m-1 \\ meets(tt_k, tt_{k+1}) = true \end{array} \right\}$$

4.5 Zeitliche Eigenschaftsattribute der ADTs

In den Definitionen der abstrakten zeitlichen Datentypen sind bereits Bedingungen enthalten, die eine konsistente zeitliche Verwaltung sicherstellen, beispielsweise die lückenlose Geschichte der Aufzeichnungszeit. Diese reichen jedoch nicht aus. So können bei der Gültigkeitszeit in einer Anwendung Überlappungen der zeitlichen Angaben erlaubt bzw. erwünscht sein und in einer anderen nicht. Diese Eigenschaft wird bei der Modellierung der Anwendung über ein zeitliches Eigenschaftsattribut des ADT explizit festgelegt und bei der Instanziierung des ADTs überprüft. Zusätzlich können Eigenschaften, wie beispielsweise ein automatisches Coalescing [Je00], eine temporale Aufwärtskompatibilität [Sn00], oder Einschränkungen des Wertebereichs, explizit über die Eigenschaftsattribute definiert werden.

4.6 Operationen der abstrakten zeitlichen Datentypen

Auf einem ADT sind alle Operationen seines ursprünglichen Datentyps sowie alle Operationen der zeitlichen Datentypen definiert. Der Unterschied zu den originalen Operationen liegt lediglich in dem neuen Ergebnistyp. Durch die Mengenwertigkeit der ADTs kann das Ergebnis mehrere Elemente enthalten, die zusammen wiederum einen instantiierten ADT bilden.

Für den Umgang der Mengenwertigkeit stehen Operationen zur Verfügung, die z.B. die Anzahl der Elemente (z.B. *count*()) oder deren zeitliche Ordnung (z.B. *last*(), *first*(), *next*()) innerhalb eines ADTs ermitteln. Zusätzlich existieren für die Änderungsrate eines Wertes (*rate_of_change*()) und dessen zeitlichen Durchschnitt (*temporal_average*()) weitere Operationen. Die Operationen *time*() und *value*() sind Projektionsoperationen die für einen ADT mit *count*()=1 die Zeitangabe oder den Wert zurückliefern. Für die Unterscheidung der Zeitarten in den bitemporalen ADTs stehen die Operationen *valid_time*() und *transaction_time*() zur Verfügung, welche für nachfolgende Operationen die zu verwendende Zeitart festlegen.
Beim Vergleich zweier abstrakter zeitlicher Datentypen *A* und *B* ruft deren mögliche Mengenwertigkeit Probleme hervor. Für den einfacheren Fall, dass entweder *A* oder *B* ein einzelnes Element *x* beinhaltet (d.h. *count*()=1), ist das Ergebnis wiederum ein ADT mit genau den Elementen, die bei dem Vergleich mit *x* das gewünschte Prädikat besitzen. Der schwierigere Fall liegt vor, wenn sowohl *A* als auch *B* jeweils mehr als ein Element besitzen. Dadurch ist es möglich, dass die einzelnen Elemente aus *A* mit jeweils unterschiedlichen Teilmengen aus *B* korrelieren. Im Ergebnis müsste somit für jedes Element aus *A* die jeweilige Ergebnismenge aus *B* dargestellt werden. Ein derartiger Vergleich liefert eine Multimenge als Ergebnis. Dieses ist mit den bisher vorgestellten Datentypen nicht darstellbar. Es kann aber, wie später noch gezeigt wird, mit Hilfe einer neuen Funktion in der Anfragesprache von T-XPath realisiert werden.

4.7 Repräsentation der abstrakten Datentypen in XML und XSchema

Die abstrakten zeitlichen Datentypen sind in XSchema integriert und können durch den Anwender analog zu den einfachen Datentypen bei einer Modellierung verwendet werden. Im Unterschied zu diesen ist ihre Struktur komplexer. In Abbildung 4.2[1] ist diese Struktur beispielhaft anhand einer ComplexTyp-Definition in XSchema dargestellt (links oben in der Abbildung). Die explizite (nicht vollständige) Definition des ADTs würde bei der Integration in XSchema wegfallen.

```
<!-- Complex Typ-Definition eines v_string -->
<xs:complexType name="v_string">
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="value" type="xs:string"/>
    <xs:element name="valid_time" type="Zeitwert"/>
  </xs:sequence>
</xs:complexType>
...
<!-- Anwendungsbeispiel eines v_string -->
<xs:complexType name="Gebäude">
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="Verwendung" type="v_string"/>
    </xs:sequence>
</xs:complexType>
```

```
...
<Gebäude>
  <Verwendung>
    <value>Stallungen</value>
    <valid_time>1500-1730</valid_time>
     <value>Hospital</value>
     <valid_time>1730-1735</valid_time>
    <value>Lagerhaus</value>
    <valid_time>1735-uc</valid_time>
  </Verwendung>
...
</Gebäude>
...
```

Abbildung 4.2: Beispiel für die Repräsentation des abstrakten Datentyp v_string in XSchema (links) und XML (rechts)

Die Abbildung 4.2 zeigt gleichzeitig die Anwendung des ADT *v_string* in einer Beispielmodellierung (linker, unterer Bereich der Abbildung), in welcher der Verwendungszweck eines Gebäudes einer zeitlichen Entwicklung unterliegt.

1. Der Type "Zeitwert" für das Element "valid_time", repräsentiert die Menge der vorgestellten atomaren zeitlichen Datentypen

Die Repräsentation des Beispiels in XML ist auf der rechten Seite der Abbildung dargestellt. Alle im Laufe der Geschichte aufgetretenen Verwendungen eines Gebäudes und deren jeweilige Gültigkeitszeiten sind vollständig in dem XML-Dokument enthalten. Die XML Repräsentation der Daten ist abwärtskompatibel zum XPath-Datenmodell. XPath könnte die zeitlichen Daten repräsentieren, allerdings mit der Einschränkung, dass keine zeitlichen Anfragen und Konsistenzbedingungen möglich bzw. überprüfbar wären.

5 Die Anfragesprache von T-XPath

T-XPath repräsentiert zeitliche Daten durch neue atomare und abstrakte zeitliche Datentypen. Die Anfragesprache von T-XPath ist für nicht-zeitliche Anfragen identisch mit der Anfragesprache X-Path. Für die Verarbeitung zeitlicher Anfragen ist sie um eine Reihe neuer zeitlicher Operationen und Funktionen (siehe Abschnitt 3.2 und Abschnitt 4.6) erweitert. Die zeitliche Unterstützung von T-XPath konzentriert sich auf die Prädikate, die mittels zeitlicher Ausdrücke eine Knotenmenge weiter verfeinern. Anhand einiger Beispiele wird im folgenden die Sprache von T-XPath näher vorgestellt.

5.1 Beispiel für Anfragen in T-XPath

Die nachfolgenden Anfragen beziehen sich auf das Beispiel von Abschnitt 4.7. Die folgende Anfrage liefert alle Gebäude deren Verwendungszweck im Jahre 1750 exakt bekannt war.

```
//Gebäude[Verwendung valid[1] '1750'= true]
```

Für den unsicheren Fall müsste die Anfrage folgendermaßen umformuliert werden:

```
//Gebäude[Verwendung valid '1750'= maybe]
```

Im Ergebnis sind ausschlieslich die Gebäude enthalten, bei denen aufgrund unscharfer Zeitangaben für das Jahr 1750 zumindest die Möglichkeit bestand, dass sie zu dieser Zeit dem entsprechenden Verwendungszweck dienten.
Alle Gebäude, die ab 1800 (bis heute oder bis zu ihrem Abriss) ihren Verwendungszweck nicht mehr geändert haben, werden mit Hilfe der nachfolgenden Anfrage ermittelt.

```
//Gebäude[Last(Verwendung) valid '1800'= true]
```

Die Funktion *Last*() findet zunächst den letzten Verwendungszweck eines Gebäudes und vergleicht anschließend, ob dieser bereits im Jahre 1800 gültig war.
Die bisher vorgestellten Anfragen wurden ausschließlich auf den zeitlichen Werten durchgeführt. Eine Anfrage, die zusätzlich konkrete Werte berücksichtigt, könnte wie folgt aussehen.

```
//Gebäude[Verwendung valid '1600' and Verwendung = 'Stallungen']
```

In der Anfrage wurde für das zeitliche Prädikat kein Wert angegeben. In diesem Fall wird *true* und *maybe* angenommen. Das Ergebnis bilden diejenigen Gebäude, die im Jahre 1600 (eventuell) als Stallungen verwendet wurden.

1. Die Operation `valid()`setzt sich aus mehreren von Allen's Operationen zusammen und vergleicht Zeitangaben ob diese mindestens ein gemeinsames Chronon besitzen.

5.2 Operationen zwischen zwei abstrakten zeitlichen Datentypen

Wie in Abschnitt 4.6 festgestellt, kann der Vergleich zweier ADTs ohne zusätzliche Unterstützung durch die Anfragesprache nicht realisiert werden. Als problematisch erweisst sich die Darstellung der Ergebnismenge, in der zu jedem Elemente aus A die entsprechende Teilmenge aus B zugeordnet ist.
Für die Realisierung führen wir zunächst die neue Funktion *expand*() ein, die als Übergabeparameter einen (instantiierten) ADT erhält. Die Operation konvertiert jedes einzelne Element des übergebenen ADTs in einen ADT mit genau einem Wert, d.h. *count*()=1, und liefert eine Sequenz derartiger ADTs als Ergebnis. Diese können analog zu Abschnitt 4.6 mit dem zweiten mengenwertigen ADT operieren.
Für die Iteration über eine oder mehrere Sequenz(en) wird in XPath 2.0 ein neuer Ausdruck, die For-Expression, eingeführt, der auch in T-XPath zur Verfügung steht. Mit Hilfe der For-Expression und der *expand*()-Funktion wird der Vergleich von zwei mengenwertigen ADTs realisiert. Abschließend ist dies anhand eines Beispiels verdeutlicht.

```
for $i in expand(//Gebäude/Verwendung)
return($i, for $j in //Gebäude[Verwendung valid valid_time($i)]
           return($j/Verwendung))
```

Die Anfrage liefert eine Sequenz, in der jedem Verwendungszweck eines Gebäudes, die zur gleichen Zeit gültigen Verwendungen der anderen Gebäude zugeordnet ist.

6 Verwandte Arbeiten

Unsere Behandlung von Unschärfen ist eine Integration und Weiterentwicklung der Ansätze wie sie in der Literatur zu zeitlichen Primitiven zu finden sind [DS98, CP01, PT01]. Auf dem Gebiet der Repräsentation und Abfrage von zeitlichen Daten innerhalb von XML wurde bisher nur wenig Forschung betrieben. So präsentierten Grandi und Mandroli ein Modell mit expliziten Tupelzeitstempeln zur Beschreibung von Gültigkeitszeiten innerhalb von XML-Dokumenten [GM99]. In dem Modell wird kein Datentypkonzept verwendet, sondern die gesamte Funktionalität von zeitlichen Anfragen beruht auf parametrisierten XSL-Stylesheets. Alle zeitlichen Operation werden als Stylesheets definiert, die das ursprüngliche XML-Dokument, unter Berücksichtigung der zeitlichen Bedingung, in ein (Ergebnis-) XML-Dokument transformieren. Es handelt sich bei diesem Vorschlag weniger um eine Integration zeitlicher Funktionalität in XML, sondern mehr um eine Modellierung einer solchen mit Hilfe von XML.
In der Arbeit von Amagasa et.al. wird eine temporale Erweiterung des XPath-Modells vorgeschlagen, in der lediglich den Kanten zwischen den Elementen ein Zeitstempel zugeordnet wird und nicht den Elementen selbst [AMU00]. Wie im vorderen Teil dieser Arbeit gezeigt, sind die zeitlichen Kanten jedoch eher für die zeitliche Verwaltung der Struktur des XML-Dokumentes geeignet. Von Dyreson werden zusätzlich zu den Kanten die Knoten mit Zeitinformation behaftet und formal in einer Erweiterung des XPath-Modells vorgestellt [Dy01]. Der Fokus liegt dabei mehr auf der Transaktionszeit und ihrer impliziten Gewinnung unter Verwendung der Änderungszeit einer XML-Datei. Ein vergleichbarer Ansatz wird von Oliboni et. al. gewählt [OQT01], wobei die Abfragefunktionaltität durch eine eigene SQL-artige Anfragesprache TS-QL realisiert ist.

7 Zusammenfassung

Wir haben gezeigt, wie zeitliche Informationen in XML verwaltet werden können. Die Grundlage bildet das XPath 2.0 Datenmodell. Dieses wurde um temporale ADTs erweitert, die neue zeitliche atomare Datentypen und Operationen kapseln und eine flexible, kalenderunabhängige Unterstützung sowohl von exakten als auch von unscharfen zeitlichen Angaben ermöglichen. Anhand einer Klassifikation von XPath-Elementen wurden die unterschiedlichen Semantiken der zeitlichen Verwaltung von Daten in T-XPath aufgezeigt. Sowohl Schemaversionisierung als auch Objektversionisierung sind möglich. Zusätzlich werden Gültigkeitszeiten und Aufzeichnungszeiten unterstützt. Die Anfragesprache von T-XPath stellt für zeitliche Anfragen eine Reihe neuer Operationen und Funktionen zur Verfügung. Dabei musste insbesondere das Problem der Multimengenwertigkeit von Ergebnissen gelöst werden.
Die Anfragesprache und die ADTs von T-XPath ermöglichen eine robuste und flexible Verwaltung von zeitlicher Information, wie sie für XML-Daten bisher nicht zur Verfügung stand. In der historischen Anwendung des GEIST Projektes wird T-XPath praktisch evaluiert.

8 Literaturverzeichnis

[Al84] Allen, J.F. Towards a General Theory of Action and Time. AI, 1984, pp. 123-154.

[AMU00] Amagasa, T., Masatoshi, Y., Uemura, S. A Data Model for Temporal XML Documents. Proc. DEXA 2000, London, 2000, pp. 334-344.

[CP01] Combi, C., Pozzi, G. HMAP - A temporal data model managing intervals with different granularities and indeterminacy from natural sentences. VLDB Journal 9, 2001, pp. 294-311.

[CR01] Chomicki, J., Revesz, P.Z. Parametric Spatiotemporal Objects. Bulletin IA*AI (Italian Association for Artificial Intelligence), Vol. 14, No.1, 2001

[DS98] Dyreson, C.E., Snodgrass, R.T. Supporting Valid-Time Indeterminacy. ACM Trans. Database Syst. 23(1), 1998, pp. 1-57.

[Dy01] Dyreson, C.E. Observing Transaction-time Semantics with TT-XPath. Proc. 2nd Int. Conf. on Web Information Systems Engineering (WISE2001), Kyoto, Japan, 2001, pp. 193-202.

[Er99] Erwig, M., Güting R.H., Schneider M., Vazirgiannis M. An Approach to Modeling and Querying Moving Objects in Databases. In GeoInformatica Vol.3, 1999

[GM99] Grandi, F., Mandreoli F. The Valid Web: it's Time to Go. TimeCenter TR-46,1999.

[Gü00] Güting, R.H., Böhlen, M.H, et.al. A Foundation for Representing and Querying Moving Objects. In ACM Trans. on Database Systems Vol. 25 No. 1, 2000, pp. 1-42.

[Je00] Jensen, C.S. Temporal Database Management. PhD thesis, Univ. of Arizona, 2000.

[Kr01] Kretschmer, U., Coors, et.al. Meeting the Spirit of History. In Proc. of the Int. Symp. on Virtual Reality, Archaeology and Cultural Heritage, VAST 2001, Greece, 2001.

[OQT01] Oliboni, B., Quintarelli, E. and Tanca, L. Temporal aspects of semi-structured data. Proc. 8th Int. Symp. on Temporal Representation and Reasoning (TIME-01), 2001.

[PT01] Pfoser, D., Tryfona, N. Capturing Fuzziness and Uncertainty of Spatiotemporal Objects. TIMECENTER Technical Report, TR-59, 2001

[Sk97] Skjellaug, B. Temporal Data: Time and Object Databases. Technical report, University Oslo, April 1997.

[Sn00] Snodgrass, R. T. Developing time-oriented database applications in SQL. Morgan Kaufmann Publishers, 2000.

[W3C02] W3C. XQuery 1.0 and XPath 2.0 Data Model (Working Draft 16.8.2002) http://www.w3.org/TR/query-datamodel/

Ein Ansatz zur Übertragung von Rangordnungen bei der Suche auf strukturierten Daten

Andreas Henrich und Günter Robbert
Universität Bayreuth, Fakultät für Mathematik und Physik,
Fachgruppe Informatik, 95440 Bayreuth
{andreas.henrich|guenter.robbert}@uni-bayreuth.de

Abstract: Ähnlichkeitsanfragen – oder allgemeiner Anfragen, die eine Rangordnung auf den Ergebnisdokumenten definieren – erlangen in Bereichen wie Multimedia oder Bioinformatik immer mehr an Bedeutung. Insbesondere im Bereich strukturierter Dokumente kommen dabei auch Anfragen vor, bei denen die Kriterien für die Rangordnung über verbundene Objekte definiert wird. So kann z.B. nach Bildern aufgrund einer Bedingung für den umgebenden Text gesucht werden. Steht nun eine Zugriffsstruktur für die Textblöcke bereit, so erscheint es vielversprechend, zunächst mit der Zugriffsstruktur ein Ranking der Textblöcke zu erstellen und dieses dann auf die Bilder zu überragen. Mit der Semantik dieser Übertragung und einem dazu anwendbaren Algorithmus beschäftigt sich die vorliegende Arbeit. Sie umfasst ferner eine Betrachtung verwandter Ansätze sowie die Präsentation experimenteller Ergebnisse.

1 Motivation

Bei klassischen Datenbankanwendungen haben Anfragen üblicherweise den Charakter harter Faktenbedingungen. Zum Beispiel wenn alle Aufträge des Kunden mit der Kundennummer 1234 gesucht werden. In anderen Anwendungsgebieten von Datenbanken, wie bei der Verwaltung multimedialer Dokumente oder in der Bioinformatik, sucht man dagegen oft nach den zu einem vage vorgegebenen Informationswunsch relevanten Dokumenten bzw. Objekten oder nach den zu einem vorgegebenen Musterobjekt ähnlichen Objekten. Hier wird ein Ordnungskriterium benötigt, das die Relevanz oder Ähnlichkeit der Objekte annähert und so ein Ranking der Objekte nach dem gewünschten Kriterium erlaubt. Im Normalfall werden dabei in der Antwort auf eine Anfrage nur die k relevantesten oder ähnlichsten Objekte erwartet.

Derartige „Ähnlichkeitsanfragen“ sind in der letzten Zeit in zahlreichen Arbeiten adressiert worden. Dabei sind einerseits Zugriffsstrukturen zur Sortierung der Daten nach einem einzelnen Ordnungskriterium und andererseits Algorithmen zur Kombination mehrerer jeweils nach einem Kriterium erstellter Rankings zu einem Gesamtranking vorgeschlagen worden – wir werden auf die wichtigsten Vorschläge hierzu in Abschnitt 3 genauer eingehen. Zugriffsstrukturen und Algorithmen, die eine gegebene Basismenge nach einem Ordnungskriterium sortieren, werden wir in dieser Arbeit als *Ranker* bezeichnen. *Ranker* arbeiten üblicherweise schrittweise nach einer *lazy evaluation* Strategie, die erlaubt, ihre

Ausgabe nach Art einer Pipe in UNIX nur so weit zu betrachten, wie dies in der konkreten Anwendung erforderlich ist. Typische Realisierungen von *Rankern* basieren auf mehrdimensionalen Zugriffsstrukturen – als Beispiele seien hier der M-tree [ZSAR98], der X-tree [BKK96] oder der LSD^h-tree [Hen98] genannt – oder invertierten Listen. Ein *Ranker* kann z.B. eingesetzt werden, um Bilder nach der Farbähnlichkeit im Hinblick auf ein Beispielbild sortiert auszugeben. Gerade bei Bildern gibt es aber neben der Farbähnlichkeit noch weitere relevante Ähnlichkeitskriterien wie die Texturähnlichkeit. Dieser Tatsache kann dadurch Rechnung getragen werden, dass man einen *Ranker* für die Farbähnlichkeit und einen *Ranker* für die Texturähnlichkeit einsetzt und die sich ergebenden Rangordnungen zu einem Gesamtranking verschmilzt. Für diese Aufgabe wurden Algorithmen wie Fagin's Algorithmus, Nosferatu oder Quick-Combine vorgeschlagen (vgl. Abschnitt 3). Wir werden Algorithmen, die mehrere Rangordnungen über Objekten der gleichen Grundgesamtheit zu einem kombinierten Ranking verschmelzen, als *Combiner* bezeichnen. Ein Aspekt, der bei diesen Ansätzen bisher aber nicht hinreichend betrachtet wurde, ist, dass gerade bei strukturierten multimedialen Dokumenten die zur Sortierung der gewünschten Objekte verwendeten Kriterien oft nicht für diese Objekte selbst sondern für mit diesen Objekten in einer bestimmten Beziehung (oder Relation) stehende Objekte definiert sind. Zwei Beispiele sollen dies verdeutlichen:

- Will man zu einem bestimmten Themengebiet relevante Bilder – ggf. auch Videos oder Audios – suchen, so kann man dieses Themengebiet häufig textuell besser beschreiben als z.B. durch ein Musterbild. Man kann nun ausnutzen, dass Bilder in strukturierten Dokumenten nicht isoliert sondern üblicherweise in einem textuellen Kontext vorkommen. Es gibt Textpassagen in der Nähe des Bildes und oft auch eine Bildunterschrift. Für diese Textobjekte im Umfeld der Bilder – was „Umfeld" im Einzelfall konkret bedeutet, ist natürlich zu spezifizieren – kann dabei mit Methoden des Information Retrieval ein Ranking nach ihrer Relevanz für das gegebene Themengebiet abgeleitet werden. Anschließend kann man versuchen, dieses Ranking auf die Bilder zu übertragen, um so die k „relevantesten" Bilder zu bestimmen.
- Ein anderes Beispiel ergibt sich, wenn wir Bilder suchen, die ein bestimmtes Logo enthalten. Hier ist es nicht sinnvoll, die vollständigen Bilder im Hinblick auf ihre Farb- oder Texturähnlichkeit mit dem gegebenen Logo zu vergleichen. Vielmehr müssen die Segmente der Bilder, die automatisch oder manuell gewonnen werden können, mit dem Logo verglichen werden. Wir erhalten so eine Ordnung auf den Bildsegmenten, die auf die Bilder selbst übertragen werden muss.

Das adressierte Problem kann damit allgemeiner formuliert werden: Gesucht werden Objekte des Typs ot_d (d für *desired*). Diese Objekte des Typs ot_d sind über eine wohldefinierte Beziehung rel_d (z.B. eine Fremdschlüsselbeziehung) mit Objekten des Typs ot_r (r für *related*) verbunden. Für die Objekte des Typs ot_r wird nun eine Relevanzordnung definiert, die auf die im Ergebnis gewünschten Objekte des Typs ot_d übertragen werden muss.

Zur Lösung dieses Problems schlagen wir vor, neben *Rankern* und *Combinern* sogenannte *Transferer* einzusetzen, die die „Übertragung" eines Rankings leisten. Es erscheint uns nützlich, diesen Transferprozess explizit zu machen und ihn nicht in den *Combinern* zu

integrieren, weil nur so die Semantik der Transferoperation und der entsprechende Algorithmus allgemeingültig angesprochen werden kann. *Ranker*, *Combiner* und *Transferer* arbeiten dabei als Produzenten und im Falle der *Combiner* und *Transferer* auch als Konsumenten von *„Objektströmen"*. Der Begriff des *Stroms* ist hier analog zur Verwendung des *Stream*-Begriffs in C++ oder Java im Sinne einer schrittweisen Erzeugung bzw. Anlieferung von Ergebnisobjekten zu verstehen. So können die initial von *Rankern* erzeugten Ströme beliebig als Eingabe für *Combiner* oder *Transferer* genutzt werden, deren Ausgaben wiederum in nachgeschaltete *Combiner* oder *Transferer* einfließen können. Dabei ist anzumerken, dass man sich neben *Rankern*, *Combinern* und *Transferern* natürlich noch weitere „Strom-verarbeitene" Komponenten vorstellen kann. Zu denken ist z.B. an Filter, die aufgrund von Faktenbedingungen aus der Ausgabe eines *Rankers*, *Combiners* oder *Transferers* die Objekte entfernen, die die spezifizierte Bedingung nicht erfüllen.

Im vorliegenden Papier betrachten wir nun den Prozess der Übertragung eines Rankings aus verschiedenen Perspektiven. Zunächst ist die genaue Semantik der Übertragung zu klären. Wie kann z.B. die Relevanzbeurteilung für Textblöcke in der Umgebung eines Bildes sinnvoll auf das Bild übertragen werden? Wir werden dieser Frage in Abschnitt 4 nachgehen. Im Anschluss daran werden wir den grundlegenden Algorithmus für die Übertragung einer Relevanzordnung in Abschnitt 5 betrachten. In Abschnitt 6 werden wir der Frage nachgehen, wie sich der vorgeschlagene Ansatz in einer Benutzungsoberfläche zur Anfrageformulierung auf strukturierten Dokumenten niederschlagen kann. Schließlich werden wir in Abschnitt 7 Überlegungen zur Systemarchitektur angeben und erste experimentelle Ergebnisse vorstellen. Zuvor soll aber in Abschnitt 2 ein einführendes Beispiel gegeben und in Abschnitt 3 auf verwandte Ansätze Bezug genommen werden.

2 Ein einführendes Beispiel

Um unseren Ansatz so genau wie möglich beschreiben zu können, führen wir zunächst ein Beispielschema ein, das im Wesentlichen auf den Vorschlägen von Analyti et al. [AC97] basiert. Abbildung 1 zeigt dieses Schema.

Im oberen linken Bereich der Abbildung ist die Struktur der Dokumente modelliert. Wir nehmen an, dass ein Multimediadokument aus einer oder mehreren Unterstrukturen besteht, die wir im Folgenden als *„Chunks"* bezeichnen wollen. Diese *Chunks* bestehen ihrerseits aus atomaren Medienobjekten oder *Chunks* einer niedrigeren Ebene. Im unteren Bereich von Abbildung 1 sind die verschiedenen Subtypen zu Medienobjekten dargestellt – nämlich *Image*, *Video*, *Audio* und *Text*. Der dritte Teil unseres Beispielschemas, der im oberen rechten Bereich zu sehen ist, repräsentiert die potentielle Segmentierung der Medienobjekte. So könnte ein Bild in die Bereiche segmentiert sein, die die konzeptuellen Objekte repräsentieren – hier sprechen wir von einer räumlichen Segmentierung – oder ein Video könnte in einzelne Szenen aufgeteilt sein – in diesem Fall sprechen wir von einer zeitlichen Segmentierung.

Ausgehend von diesem Schema können wir eine Anfrage formulieren, die die in der Einleitung angeführten Szenarien aufnimmt und als typisches Anwendungsbeispiel für den in

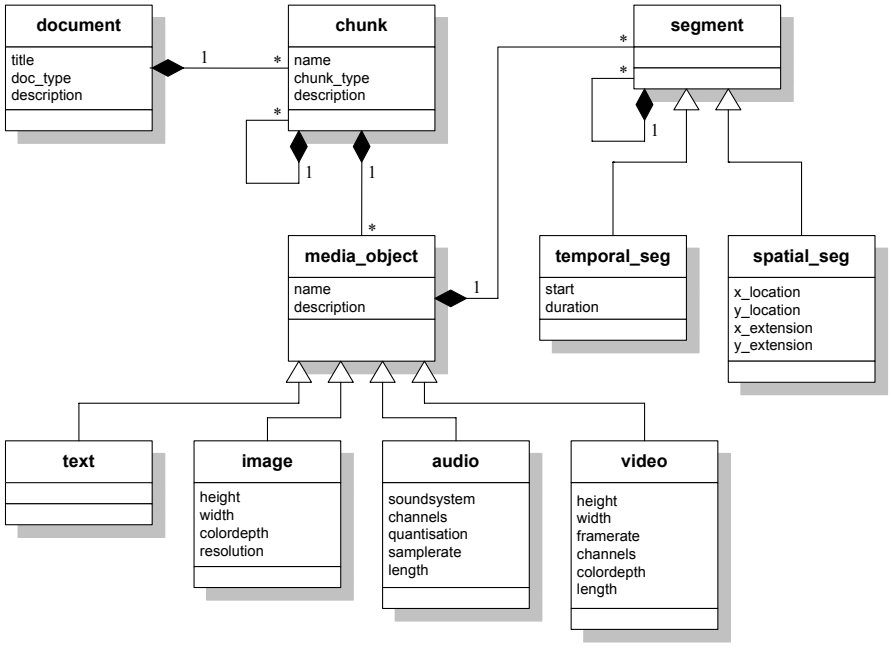


Abbildung 1: Ein Beispielschema für multimediale Dokumente

Abschnitt 5 vorgestellten *RSV-Transfer*-Algorithmus angesehen werden kann. Wir gehen dazu von einem Anwender aus, der nach Bildern sucht, die ein bestimmtes Logo enthalten und in deren Nähe der Text von *Skifahren* oder allgemeiner von *Wintersport* handelt. In diesem Fall sind die gewünschten Objekte Bilder. Zwei Ordnungskriterien sind definiert: (1) *Das Bild soll ein gegebenes Logo enthalten.* Diese Bedingung kann als Ähnlichkeitsbedingung für die dem Bild zugeordneten Segmente formuliert werden. Wir suchen daher nach einem Bild, für das eines der zugeordneten Bildsegmente im Hinblick auf Farbe, Textur und ggf. Form ähnlich zu dem vorgegebenen Logo ist. (2) *Der Text in der Umgebung des Bildes soll sich mit „Skifahren“ oder allgemeiner „Wintersport“ beschäftigen.* Ausgehend von unserem Beispielschema können wir den Text in der Nähe eines Bildes als die Menge der Textobjekte definieren, die im gleichen *Chunk* wie das Bild enthalten sind. Wir können dabei z.B. das Vektorraummodell [Sal89] verwenden, um diese Textobjekte im Hinblick auf ihre Ähnlichkeit zum Anfragetext „Skifahren oder Wintersport“ zu ordnen.

Durch diese beiden Ordnungskriterien haben wir nun ein Ranking der Bildsegmente und ein Ranking der Textobjekte und wir müssen aus diesen beiden Rangordnungen ein Ranking für die gesuchten Bildobjekte selbst ableiten. Abbildung 2 verdeutlicht, wie dies mit Hilfe von *Rankern*, *Combinern* und *Tranferern* geschehen kann.

Zunächst werden in Schritt (1.) mit Hilfe entsprechender *Ranker* vier initiale Ordnungen erstellt. Wir gehen dabei davon aus, dass für die Bildsegmente im Hinblick auf die Ähnlichkeit zum gegebenen Logo drei unterschiedliche Ähnlichkeitskriterien angewendet werden. Die drei sich hieraus ergebenden Rankings werden in Schritt (2.) zu einem ge-

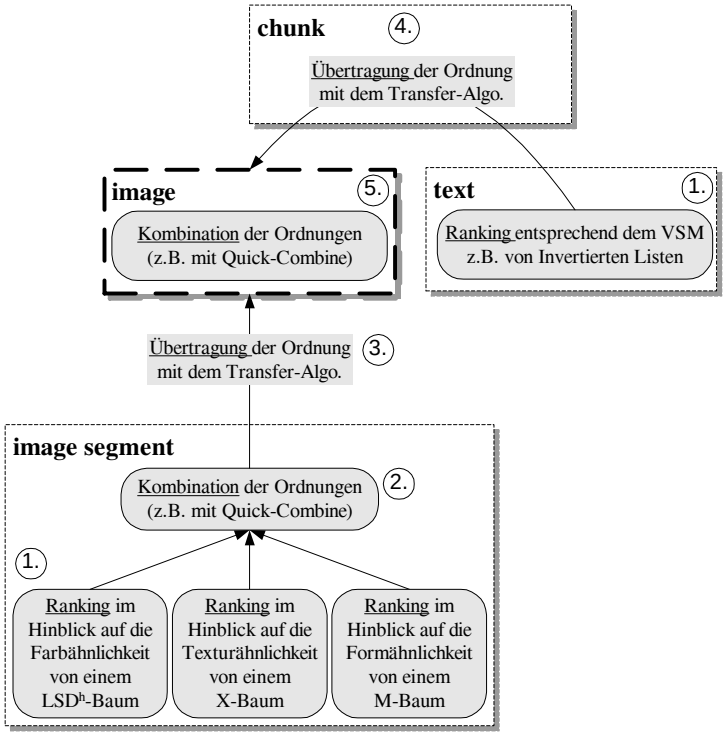


Abbildung 2: Ablauf der Bearbeitung der Beispielanfrage

meinsamen Ranking für die Bildsegmente kombiniert. Damit liegen die Rangordnungen für die Textdokumente und die Bildsegmente vor. Diese Ordnungen werden in den Schritten (3.) und (4.) mit dem *RSV-Transfer*-Algorithmus auf die Bilder übertragen. Schließlich werden in Schritt (5.) die beiden sich hieraus ergebenden Rangordnungen für die Bilder kombiniert.

An dieser Stelle sei angemerkt, dass man sich im obigen Beispiel natürlich auch eine speziell auf diese Anfrage zugeschnittene Zugriffsstruktur vorstellen kann. Eine solche „Speziallösung“ scheitert aber zwangsweise schon an geringfügig modifizierten Anfragen. Unser Ansatz ist dagegen durch die Kombination von *Rankern*, *Combinern* und *Transferern* flexibel auf ein breites Spektrum von Anfragen anwendbar.

3 Ein Überblick über Verfahren der Ähnlichkeitssuche

Wie bereits in der Motivation erwähnt, wurden in den letzten Jahren zum Thema Ähnlichkeitssuche eine Vielzahl von Artikeln publiziert. Die hinsichtlich unseres Ansatzes relevante Literatur lässt sich in die folgenden Kategorien unterteilen:

Verfahren zur Berechnung von Ähnlichkeitsanfragen

Im Rahmen dieses Forschungsgebietes wurden und werden Verfahren entwickelt, die als Ergebnis einer Ähnlichkeitsanfrage eine anhand eines vorgegebenen Ordnungskriteriums sortierte Liste (eine Rangordnung) von Elementen zurückliefern. Hierbei interessieren den Informationssuchenden meist nur die ersten k relevanten Treffer des Ergebnisses. Deshalb beschäftigt sich ein Forschungszweig explizit mit dem Thema der möglichst effizienten Berechnung der ersten k-Treffer von Ähnlichkeitsanfragen (Top-k Anfragen). Hierbei werden zwei unterschiedliche Aspekte betrachtet:

Im ersten Bereich wird das Problem aus der Perspektive der Anfrageoptimierung unter anderem in objektrelationalen Datenbanken betrachtet. In den Arbeiten von Carey und Kossmanns [CK98, CK97] wird aufgezeigt, wie ein STOP-AFTER-Operator, der zur Angabe der Kardinalität des Ergebnisses bei der Anfrageformulierung dient, in ein Datenbanksystem integriert werden kann. Insbesondere wird dargelegt, welche Strategien bei der internen Anfragebearbeitung zum Einsatz kommen können, um Anfragen unter Verwendung des STOP-AFTER-Operators effizient abarbeiten zu können. Eine andere Möglichkeit zur Optimierung wird in den Artikeln von Chaudhuri und Gravano [BCG02] beschrieben. Hier wird basierend auf Datenbankstatistiken versucht eine Top-k Anfrage mittels geeigneter Anfrageumformulierungen so auf komplexe Bereichsanfragen auf ein Datenbanksystem abzubilden, dass nur ein möglichst kleiner Teil des Datenbestandes zur Berechnung des Anfrageergebnisses herangezogen werden muss.

Im zweiten Bereich geht es um die Entwicklung spezieller Zugriffs- oder Indexstrukturen, welche Ähnlichkeitsanfragen effizient unterstützen. Für diesen Zweck wurden zahlreiche Ansätze und Implementierungen vorgestellt. Häufig wurden hierzu mehrdimensionale Bäume eingesetzt, wie beispielsweise der M-tree [ZSAR98], der X-tree [BKK96] oder der LSD^h-tree [Hen98]. Andere Realisierungen verwenden als Indexstruktur invertierte Listen [SMMR99] oder basieren auf einem schnellen sequentiellen Durchlauf der verwalteten Daten, wie z.B. bei den VA-Files [WSB98]. Im Sinne unseres Sprachgebrauchs handelt es sich hier um mögliche Implementierungen von *Rankern*.

Verfahren zur Kombination von Rangordnungen

In der Motivation haben wir aufgezeigt, dass es Anwendungsfälle gibt, in denen die Kombination mehrerer Rangordnungen zu einer Gesamtordnung notwendig ist. Hierzu wurden effiziente Verfahren zur Kombination von Rangordnungen entwickelt. Diese Verfahren gewinnen in letzter Zeit stark an Bedeutung, da sie nicht nur bei der Verschmelzung mehrerer Rangordnungen, die von einem einzelnen Datenbanksystem geliefert werden, anwendbar sind. Vielmehr finden solche Algorithmen zunehmend auch bei der Berechnung von Gesamtergebnissen über verteilte Datenkollektionen hinweg Anwendung. Problematisch sind dabei aber – insbesondere wenn Rangordnungen von heterogenen Systemen kombiniert werden sollen – die unterschiedlichen Zugriffsmöglichkeiten auf die einzelnen Rangordnungen, die die Datenbanksysteme anbieten, sowie die Ermittlung des Gesamtrankings. Diese Problematik wurde schon vor einigen Jahren erkannt und als *Collection-Fusion-Problem* bezeichnet [VGJL94].

Zwischenzeitlich wurde eine Vielzahl von Kombinationsalgorithmen publiziert, die vom Arbeitsprinzip her ähnlich vorgehen, sich aber durch unterschiedliche Anforderungen und

Anforderung/ Verfahren	wahlfreier Zugriff	sortierter Zugriff	inkrementell anwendbar	gleiche Objekt-IDs
Buckley-Lewitt	nicht notwendig	nicht notwendig	nein	ja
FA	erforderlich	erforderlich	nein	ja
TA	erforderlich	erforderlich	nein	ja
NRA	nicht notwendig	erforderlich	nein	ja
NRA-RJ	nicht notwendig	erforderlich	ja	ja
J*	nicht notwendig	erforderlich	ja	nein
Nosferatu	nicht notwendig	erforderlich	ja	ja
Quick-Combine	erforderlich	erforderlich	nein	ja
Stream-Combine	nicht notwendig	erforderlich	ja	ja
Rank-Combine	nicht notwendig	erforderlich	ja	ja

Tabelle 1: Anforderungen und Eigenschaften der jeweiligen Kombinationsverfahren

Eigenschaften auszeichnen. In Tabelle 1 werden die wesentlichen Eigenschaften und Anforderungen der im Weiteren aufgeführten Verfahren gegenübergestellt. Wir untersuchen hierbei, welche Arten des Zugriffs von den eingehenden Rangordnungen unterstützt werden müssen, damit die Kombinationsalgortihmen anwendbar sind. Wir unterscheiden zwischen zwei Arten des Zugriffs, dem sortierten und dem wahlfreien Zugriff. Als sortierter Zugriff wird ein Zugriff bezeichnet, bei dem nacheinander in sortierter Reihenfolge Elemente aus einer Rangordnung entnommen werden. Von einem wahlfreien Zugriff sprechen wir, wenn zu einer vorgegebenen Objekt-ID der so genannte *Retrieval-Status-Wert* (vgl. Abschnitt 4) des zugehörigen Objekts in einer Rangordnung gesucht wird. Des weiteren wird in Tabelle 1 aufgezeigt, ob ein Verfahren inkrementell eingesetzt werden kann, d.h. ob weitere Verarbeitungsschritte auf höherer Ebene erfolgen können bevor die Berechnung der Gesamtliste abgeschlossen ist. Dies hat zur Folge, dass bei inkrementellen Verfahren der Wert k nicht vor Beginn der Berechnung bekannt sein muss.

Als weitere Eigenschaft untersuchen wir, ob ein Algorithmus die Kombination von Rangordnungen auch dann ermöglicht, wenn die in den zu kombinierenden Ordnungen enthalten Objekt-IDs disjunkt sind. Hierbei zeigt sich, dass viele Verfahren nur dann einsetzbar sind, wenn die Kombination über nicht disjunkte Mengen von Objekt-IDs erfolgen kann. Dies hat zur Folge, dass solche Verfahren Gesamtordnungen nur bestimmen können, wenn in den Eingangsordnungen die gleichen Objekt-IDs enthalten sind, wobei die Objekt-IDs natürlich an unterschiedlichen Stellen der verschiedenen Ordnungen auftreten können.

Einer der ersten Algorithmen zur Kombination von Rangordnungen wurde von Buckley und Lewit [BL85] im Zusammenhang mit invertierten Listen publiziert. Dieser Algorithmus benötigt weder einen wahlfreien noch einen sortierten Zugriff auf die Eingangslisten sondern lediglich einen Zugriff auf die Listenelemente in beliebiger Folge. Er ist nur sehr beschränkt inkrementell anwendbar. Grundlegende Arbeiten zum Thema Kombination von Rangordnungen wurden von Fagin geleistet, der mit dem Algorithmus FA (Fagin's Algorithm) [Fag98] ein effizientes Kombinationsverfahren vorstellte. Dieser Algorithmus benötigt für die Verarbeitung sowohl den sortierten als auch den wahlfreien Zugriff. Später folgte eine verbesserte Version von FA, TA (Threshold Algorithm) [FLN01]

genannt. TA verwendet ebenfalls den sortierten und den wahlfreien Zugriff für die Berechnung der Gesamtordnung. Sowohl FA als auch TA können nicht inkrementell angewendet werden. Die Algorithmen Quick-Combine [GBK00] und Multi-Step [NR99] basieren auf Fagin´s Algorithmus TA und unterscheiden sich im Wesentlichen durch verbesserte Abbruchbedingungen. Im Gegensatz zu den genannten Algorithmen benötigen die Verfahren NRA [Fag98] und Nosferatu [PP97] keinen wahlfreien Zugriff, NRA ist in der publizierten Originalversion im Vergleich zu Nosferatu jedoch nicht inkrementell einsetzbar. Der Algorihtmus NRA-RJ [IAE02] stellt eine Verbesserung des NRA-Algorithmus dar, der inkrementell eingesetzt werden kann. Weitere interessante Algorithmen für die Kombination von Rangordnungen sind RankCombine [HR01b], StreamCombine [GBK01], und J* [NCS$^+$01]. Alle drei Verfahren benötigen keinen wahlfreien Zugriff und sind zudem inkrementell einsetzbar. Nur der J* Algorithmus unterstützt die Kombination von Rangordnungen im Falle disjunkter Objekt-IDs. Mehr noch erlaubt J* die Übertragung von Ordnungen eines Objekttyps auf einen anderen Objekttyp, da hier mit Hilfe von benutzerdefinierten Prädikaten gesteuert werden kann, wie die verschiedenen Listen zu kombinieren sind. Die Übertragung der Rangordnungen erfolgt beim J* Algorithmus jedoch implizit bei der Kombination und weder die Semantik noch der Zusammenhang zwischen Kombination und Übertragung werden von den Autoren angesprochen. Im Gegensatz dazu verfolgen wir in dieser Arbeit den Ansatz, die Übertragung einer Rangordnung explizit zu betrachten und von ggf. vor- oder nachgelagerten Kombinationen von Rangordnungen zu trennen.

4 Die Semantik der Übertragung einer Ordnung

Bevor wir in Abschnitt 5 den Algorithmus zur Übertragung einer Rangordnung von einem Objekttyp auf einen verbundenen Objekttyp vorstellen, müssen wir zunächst die denkbaren Semantiken für diese Übertragung betrachten. Hierzu greifen wir auf die in Abschnitt 2 eingeführte Beispielanfrage zurück und vereinfachen sie zunächst indem wir nur die Suche nach Bildern betrachten, die ein gegebenes Logo enthalten. Die Situation ist hier wie folgt: Zunächst wird für die Bildsegmente eine Rangordnung berechnet. Dazu wird für die einzelnen Bildsegmente ein so genannter *Retrieval-Status-Wert* berechnet, der sich z.B. aus dem Vergleich der Farbhistogramme des Bildsegmentes und des Anfragelogos oder durch einen *Combiner* als gewichtete Kombination der Farb-, der Textur- und ggf. der Formähnlichkeit ergeben kann. Diese Retrieval-Status-Werte definieren nun zwar eine Rangordnung auf den Bildsegmenten, wir sind aber an einem Ranking der Bilder selbst interessiert. Daher ergibt sich die Notwendigkeit, für jedes Bild einen abgeleiteten Retrieval-Status-Wert auf Basis der Retrieval-Status-Werte der ihm zugeordneten Bildsegmente zu berechnen, um die gewünschte Ordnung auf den Bildern zu definieren.

$RSV_r(ro)$ sei der Retrieval-Status-Wert (*retrieval status value*) des Objekts ro (ro für „related object“ und RSV_r für den RSV eines verbundenen (*related*) Objekts). In unserem Beispiel wäre ro ein Bildsegment. Des weiteren sei $\{ro_{i,1}, ro_{i,2}, \ldots, ro_{i,n_i}\}$ die Menge der verbundenen Objekte die mit dem gewünschten (*desired*) Objekt do_i in Beziehung stehen. In unserem Beispiel würde diese Menge alle Bildsegmente enthalten, die dem Bild

do_i zugeordnet sind. Schließlich wollen wir annehmen, dass hohe Retrieval-Status-Werte besonders relevante Objekte identifizieren. Dann benötigen wir eine Funktion $\mathcal{F}$, die den abgeleiteten Retrieval-Status-Wert $RSV_d(do_i)$ von den mit do_i verbundenen Objekten und ihren Retrieval-Status-Werten ableitet:

$$RSV_d(do_i) \stackrel{\text{def}}{=} \mathcal{F}\,(\ \langle ro_{i,1}, RSV_r(ro_{i,1})\rangle,\ \ldots,\ \langle ro_{i,n_i}, RSV_r(ro_{i,n_i})\rangle\ )$$

Wir wollen nun einige für die Praxis relevante Beispiele für die Funktion $\mathcal{F}$ betrachten:

- *Der maximale RSV_r Wert*

  In diesem Fall vereinfacht sich die Funktion zu $RSV_d(do_i) \stackrel{\text{def}}{=} \max\{RSV_r(ro_{i,1}), \ldots, RSV_r(ro_{i,n_i})\}$. Dies bedeutet, dass der Retrieval-Status-Wert des gewünschten Objekts do_i durch das verbundene Objekt mit dem höchsten Retrieval-Status-Wert definiert wird. In unserem Beispiel würde dies dazu führen, dass das dem Logo ähnlichste Bildsegment den Retrieval-Status-Wert des gesamten Bildes definiert.

  Nun muss noch die Berechnung für Objekte geklärt werden, zu denen keine verbundenen Objekte existieren. In diesem Fall sollte $RSV_d(do_i)$ zum kleinsten denkbaren Retrieval-Status-Wert definiert werden – typischerweise null. Dies ist auch für die folgenden alternativen Definitionen der Funktion $\mathcal{F}$ eine sinnvolle Wahl.

- *Der durchschnittliche RSV_r Wert*

  Der Durchschnitt über alle RSV_r Werte ist ein anderes Beispiel für eine mögliche Semantik: $RSV_d(do_i) \stackrel{\text{def}}{=} \frac{1}{n_i} \cdot \sum_{j=1}^{n_i} RSV_r(ro_{i,j})$. Eine nähere Betrachtung ergibt allerdings, dass die Bedeutung des Durchschnitts stark vom angewendeten Ähnlichkeitsmodell abhängt. Im Hinblick auf unser Beispiel könnte der Retrieval-Status-Wert eines Bildsegmentes verglichen mit dem gegebenen Logo zum Beispiel über eine normierte Farbähnlichkeit definiert sein. Die Normierung würde dazu führen, dass Segmente unterschiedlicher Größe trotzdem den gleichen Einfluss auf die Ähnlichkeitssortierung der Bilder haben. Analoge Situationen können auftreten, wenn verbundene Textobjekte adressiert werden und das Vektorraummodell mit einer Dokumentlängennormierung angewendet wird. In diesen Fällen erscheint daher die Anwendung eines „gewichteten Durchschnitts“ ggf. angemessener.

- *Ein gewichteter Durchschnitt*

  Um einen gewichteten Durchschnitt zu berechnen, benötigen wir eine Definition der Größe (size) eines Objekts ro_i. Für Bildsegmente könnte diese Größe über die Zahl der Pixel und für Textobjekte über die Zahl der Worte definiert werden. Ausgehend von einer solchen Größendefinition können wir folgende Semantik definieren:

  $$RSV_d(do_i) \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} RSV_r(ro_{i,j}) \cdot \frac{\mathsf{size}(ro_{i,j})}{\sum_{h=1}^{n_i} \mathsf{size}(ro_{i,h})}$$

  Für einen breiten Bereich von Ähnlichkeitskriterien bedeutet diese Definition, dass letztlich die Vereinigung über alle verbundenen Objekte $\left(\bigcup_{j=1}^{n_i} ro_{i,j}\right)$ betrachtet

wird, um den Retrieval-Status-Wert für do_i zu berechnen. Für Bilder mit verbundenen Bildsegmenten bedeutet dies, dass das Bild als Vereinigung der verbundenen Segmente betrachtet wird. Für einen Abschnitt mit Textobjekten bedeutet es, dass der Abschnitt als Konkatenation der verbundenen Textobjekte betrachtet wird.

- *Der minimale RSV_r Wert*

  In einigen Situationen kann es auch sinnvoll sein, die Ähnlichkeit des gewünschten Objekts über das „unähnlichste" verbundene Objekt zu definieren. In diesem Fall ergibt sich $RSV_d(do_i) \stackrel{\text{def}}{=} \min\{RSV_r(ro_{i,1}), \ldots, RSV_r(ro_{i,n_i})\}$. Hier wird der Retrieval-Status-Wert für do_i über die „am schlechtesten passende" Komponente bestimmt. Ein Beispiel, in dem dies sinnvoll sein könnte, ergibt sich, wenn im Hinblick auf die verbundenen Objekte sehr homogene Objekte do_i gesucht werden.

Neben den oben angeführten Semantiken existieren weitere Möglichkeiten. So könnte der Median oder ein α-Perzentil in entsprechenden Situationen sinnvoll sein. Im Folgenden werden wir sehen, dass unser Ansatz für die Übertragung einer Ordnung auch bei solchen Definitionen anwendbar bleibt, sofern die Semantik sicherstellt, dass $RSV_d(do_i) \leq \max\{RSV_r(ro_{i,1}), \ldots, RSV_r(ro_{i,n_i})\}$ gilt. Für $\mathcal{F}$ scheiden damit lediglich Funktionen wie die Summe oder das Produkt aus. Ob der Einsatz solcher Aggregationsfunktionen im Rahmen von Ähnlichkeitsanfragen sinnvolle Anwendungen hat, ist derzeit nicht klar und Gegenstand unserer aktuellen Forschungsarbeiten.

5 Der RSV-Transfer Algorithmus

Nachdem wir die Semantik der Übertragung eines Ähnlichkeitskriteriums betrachtet haben, können wir uns dem entsprechenden Algorithmus zuwenden. Das Problem, welches bei der Übertragung einer Rangordnung gelöst werden muss, kann wie folgt beschrieben werden: Wir betrachten eine Anfrage, die eine Ordnung für Objekte eines gewünschten Typs ot_d fordert (z.B. für Bilder). Allerdings ist die Ordnung nicht für die Objekte des Typs ot_d definiert, sondern für verbundene Objekte des Typs ot_r (z.B. Bildsegmente).

Wir nehmen nun an, dass die *„Verbindung"* (oder *„Relation"*) zwischen diesen Objekten wohldefiniert ist (z.B. durch einen Pfadausdruck) und dass sie bidirektional traversiert werden kann. Dies bedeutet, dass wir das betroffene Objekt – oder die betroffenen Objekte – des Typs ot_d für ein Objekt vom Typ ot_r und die verbundenen Objekte vom Typ ot_r für ein Objekt vom Typ ot_d bestimmen können. In unserem Beispiel bedeutet dies, dass wir für jedes Bildsegment sein Ursprungsbild und für jedes Bild die zugeordneten Bildsegmente (effizient) ermitteln können. In unserem konkreten Beispiel wird es nur ein Ursprungsbild für jedes Bildsegment geben. Es sind aber Situationen denkbar, in denen ein verbundenes Objekt mehreren Objekten des Typs ot_d zugeordnet ist. Die konkreten Charakteristika der Traversierungsoperationen in beiden Richtungen hängen offensichtlich von der Datenbank oder dem Objektspeicher ab, der zur Verwaltung der Dokumente eingesetzt wird. Bei objektrelationalen Datenbanken erlauben z.B. Join-Indizes und Indexstrukturen für Nested Tables ein hinreichend effizientes Traversieren der Verbindungen.

Zusätzlich nehmen wir an, dass eine Eingabeordnung (z.B. aus einem *Ranker* oder *Combiner*) gegeben ist, die eine Sortierung der Objekte des Typs ot_r liefert.

Ausgehend von diesen Annahmen, kann der Übertragungsalgorithmus wie folgt arbeiten: Er benutzt die Rangordnung – bzw. in Anlehnung an die Begrifflichkeiten von C++ oder Java den *Strom* – mit den geordneten Objekten vom Typ ot_r als Eingabe. Für die Elemente, die schrittweise aus diesem Eingabestrom entnommen werden, wird das betroffene Objekt – oder die betroffenen Objekte – vom Typ ot_d durch eine Traversierung der entsprechenden Beziehungen ermittelt. Nun werden die RSV_d Werte für diese Objekte vom Typ ot_d entsprechend der gewählten Semantik bestimmt und das aktuell betrachtete Objekt vom Typ ot_d wird in eine Hilfsdatenstruktur eingefügt, in der alle bisher betrachteten Objekte gemeinsam mit ihren RSV_d Werten verwaltet werden. Nun wird das nächste Objekt vom Typ ot_r aus dem Eingabestrom entnommen und untersucht; ist der RSV_r Wert dieses Objekts kleiner als der höchste RSV_d Wert eines Elements in der Hilfsdatenstruktur, das noch nicht ausgegeben wurde, so wird dieses Element aus der Hilfsdatenstruktur nun im Ausgabestrom des Übertragungsalgorithmus ausgegeben.

Für eine detailliertere Betrachtung des Algorithmus müssen wir die Charakteristika der Hilfsdatenstruktur festlegen, die wir mit AL (*auxiliary list*) bezeichnen wollen. AL verwaltet die vorläufigen Ergebnisobjekte zusammen mit ihren RSV_d Werten. Genauer verwaltet AL Paare der Form $\langle do_i; RSV_d(do_i)\rangle$ mit type$(do_i) = ot_d$. Diese Paare sind nach den RSV_d Werten absteigend sortiert. Für die Hilfsdatenstruktur AL werden nun die folgenden Operationen benötigt: createAL() erzeugt eine leere Hilfsdatenstruktur. getObj(AL, i) liefert das Objekt, dessen RSV_d Wert unter den Objekten in AL Rang i einnimmt. getRSV(AL, i) liefert den RSV_d Wert für das Objekt, dessen RSV_d Wert unter den Objekten in AL Rang i einnimmt. contains(AL, do_j) prüft, ob es in AL einen Eintrag für do_j gibt. insert$(AL, \langle do_l; RSV_d(do_l)\rangle)$ fügt einen Eintrag für do_l in AL ein. Dabei wird die Sortierung im Hinblick auf die RSV_d Werte erhalten. Existieren mehrere Objekte mit dem gleichen RSV_d Wert, so wird der neue Eintrag hinter die anderen Einträge mit gleichem Wert eingereiht. size(AL) liefert die Anzahl der Einträge in AL.

Ausgehend von diesen Definitionen können wir eine Klasse *Transferer* angeben, die einen Konstruktor und eine getNext-Methode zur Verfügung stellt. Diese Klasse ist in einer programmiersprachlichen Notation in Abbildung 3 angegeben.

Die Attribute, die für ein *Transferer*-Objekt verwaltet werden müssen, umfassen den Eingabestrom, eine Definition der gewünschten Beziehung zwischen den Objekten des Typs ot_d und ot_r, die Hilfsdatenstruktur AL, eine Variable o_r, die das Element des Eingabestroms aufnimmt, welches als nächstes zu bearbeiten ist und die Anzahl der bisher in den Ausgabestrom gestellten Objekte. Der Konstruktor initialisiert diese Variablen und liest das erste Objekt aus dem Eingabestrom. Dieses Objekt wird in der Variable o_r abgelegt. Die getNext-Methode arbeitet wie folgt:

- So lange der Eingabestrom nicht vollständig abgearbeitet ist ($o_r \neq \perp$) und entweder alle aktuell in AL verwalteten Objekte bereits ausgegeben wurden (size$(AL) < k$) oder der RSV_r Wert von o_r größer als der größte RSV_d Wert eines noch nicht ausgegebenen Objekts in *AL* ist, wird das aus dem Eingabestrom stammende Objekt o_r betrachtet und das nächste Element aus dem Eingabestrom nach o_r gelesen.

```
Class Transferer {
  Stream : inputStream;
  RelationshipDef : rel_d; /* Beziehung zu den verbundenen Objekten */
  AuxiliaryList : AL;
  InputObject : o_r; /* das nächste Objekt, das betrachtet werden müßte */
  Integer : k; /* Nummer des nächsten in der Ausgabe zu liefernden Objekts */

  constructor(Stream : input, RelationshipDef : rel) {
    inputStream := input;  rel_d := rel;
    AL := createAL();
    o_r := streamGetNext(inputStream);
    if o_r = ⊥ then exception(„leerer Eingabestrom“);
    k := 1;
  }

  getNext() : OutputObject {
    while o_r ≠ ⊥ ∧ (size(AL) < k ∨ RSV_r(o_r) ≥ getRSV(AL, k)) do
      /* betrachte das nächste Objekt von der Eingabe (also o_r) */
      SDO := {o_d | ∃rel_d(o_d → o_r)};
        /* alle Objekte, die zu o_r in der gewünschten Beziehung stehen */
      foreach o_d ∈ SDO do
        if ¬contains(AL, o_d) then insert (AL, ⟨o_d; RSV_d(o_d)⟩);
          /* RSV_d muss dazu nach der gewünschten Semantik berechnet werden */
      end /* foreach */;
      o_r := streamGetNext(inputStream);
    end /* while */;
    if o_r = ⊥ ∧ size(AL) < k then
      return ⊥; /* der Eingabestrom ist vollständig abgearbeitet */
    else
      k++;  return getObj(AL, k − 1);
    end /* if */;
  }
}
```

Abbildung 3: Die Klasse *Transferer* in einer programmiersprachlichen Notation

Das Objekt o_r „zu betrachten“, bedeutet konkret die Menge SDO zu bestimmen, die die Objekte vom Typ ot_d enthält, die zu o_r in der gewünschten Beziehung stehen (SDO für „set of desired objects“). In Abbildung 3 wird hierzu die Notation $\exists rel_d(o_d \rightarrow o_r)$ verwendet, die – da wir von o_r ausgehen – deutlich macht, dass die gewünschte Beziehung hier in umgekehrter Richtung zu traversieren ist.

Für jedes Objekt o_d aus SDO, das nicht bereits in AL vorhanden ist, wird der Wert $RSV_d(o_d)$ berechnet und ein Eintrag in AL eingefügt. Die Berechnung von $RSV_d(o_d)$ erfordert dabei mit Ausnahme der Maximumsemantik – die einen Spezi-

alfall darstellt, den wir später genauer betrachten werden –, dass wir wie folgt vorgehen: (1) Die Menge mit allen verbundenen Objekten vom Typ ot_r muss bestimmt werden. (2) Für diese verbundenen Objekte müssen die RSV_r Werte berechnet werden. (3) Nun kann $RSV_d(o_d)$ von diesen RSV_r Werten abgeleitet werden, indem die der gewünschten Semantik entsprechende Funktion $\mathcal{F}$ angewendet wird.

- Falls der Eingabestrom vollständig abgearbeitet ist ($o_r = \perp$) und alle Objekte aus *AL* bereits ausgegeben wurden ($\mathsf{size}(AL) < k$), ist auch der Ausgabestrom des *Transferers* vollständig erstellt und ein „$\perp$“-Objekt wird zurückgegeben.

- Anderenfalls ist entweder der Eingabestrom vollständig abgearbeitet oder das nächste zu bearbeitende Element aus dem Eingabestrom (o_r) hat einen RSV_r Wert der höchstens so groß ist wie der höchste RSV_d Wert eines noch nicht ausgegebenen Elements in AL (durch die Sortierung in *AL* ist dies Element k in *AL*). In diesem Fall können wir dieses Element von *AL* auf dem Ausgabestrom ausgeben.

  Solange die festgelegte Semantik für RSV_d sicherstellt, dass $RSV_d(do_i) \leq \max\{RSV_r(ro_{i,1}), \ldots, RSV_r(ro_{i,n_i})\}$ gilt, arbeitet das obige Vorgehen korrekt. Diese Bedingung ist offensichtlich für die Maximum-, die Minimum-, die Durchschnitts- und die gewichtete Durchschnittssemantik erfüllt, weil diese Werte den Maximalwert der betrachteten Menge sicher nicht übersteigen können. Auf Basis der obigen Ungleichung können wir, sobald $RSV_r(o_r) < \mathsf{getRSV}(AL, k)$ gilt, sicher sein, dass das k-te Element in AL im weiteren Verlauf des Algorithmus nicht mehr von dieser Position verdrängt wird. Da neue Einträge in *AL* immer hinter bereits vorhandenen Einträgen mit gleichem RSV_d Wert abgelegt werden, gilt dies auch für $RSV_r(o_r) \leq \mathsf{getRSV}(AL, k)$.

  Dies kann wie folgt verdeutlicht werden: Falls o_r nicht das Objekt mit dem höchsten RSV_r Wert ist, das zu einem bestimmten Objekt o_d in Beziehung steht, so wurde o_d bereits betrachtet als das verbundene Objekt mit dem höchsten RSV_r Wert aus dem Eingabestrom entnommen und verarbeitet wurde. Folglich ist, wenn ein Objekt o_d zum ersten Mal betrachtet wird, das verbundene Objekt o_r, das die Betrachtung verursacht, sicher das verbundene Objekt mit dem höchsten RSV_r Wert. Durch die oben angegebene Ungleichung ist dabei sichergestellt, dass der Wert $RSV_d(o_d)$ kleiner oder gleich $RSV_r(o_r)$ ist. Daher können wir das k-te Objekt aus AL auf dem Ausgabestrom ausgeben, sobald $RSV_r(o_r) \leq \mathsf{getRSV}(AL, k)$ gilt, weil in dieser Situation AL sicher die k Objekte mit den höchsten RSV_d Werten enthält.

Wie oben erwähnt, bildet die Maximumsemantik einen Spezialfall, der einige Vereinfachungen erlaubt. Bei dieser Semantik entfällt die Notwendigkeit die RSV_d Werte in der **foreach** Schleife zu berechnen, weil immer dann, wenn noch kein Eintrag für o_d in AL vorhanden ist, o_r sicher das verbundene Objekt mit dem höchsten RSV_r Wert für o_d ist. Folglich gilt $RSV_d(o_d) = RSV_r(o_r)$. Dies hat wiederum zur Folge, dass die Operation $\mathsf{insert}\,(AL, \langle o_d; RSV_d(o_d)\rangle)$ in der getNext-Methode durch die wesentlich effizientere Operation $\mathsf{insert}\,(AL, \langle o_d; RSV_r(o_r)\rangle)$ ersetzt werden kann.

Weitere Vereinfachungsmöglichkeiten ergeben sich, wenn die Beziehung rel_d eine 1:n-Beziehung ist. In diesem Fall ist *SDO* immer eine einelementige Menge.

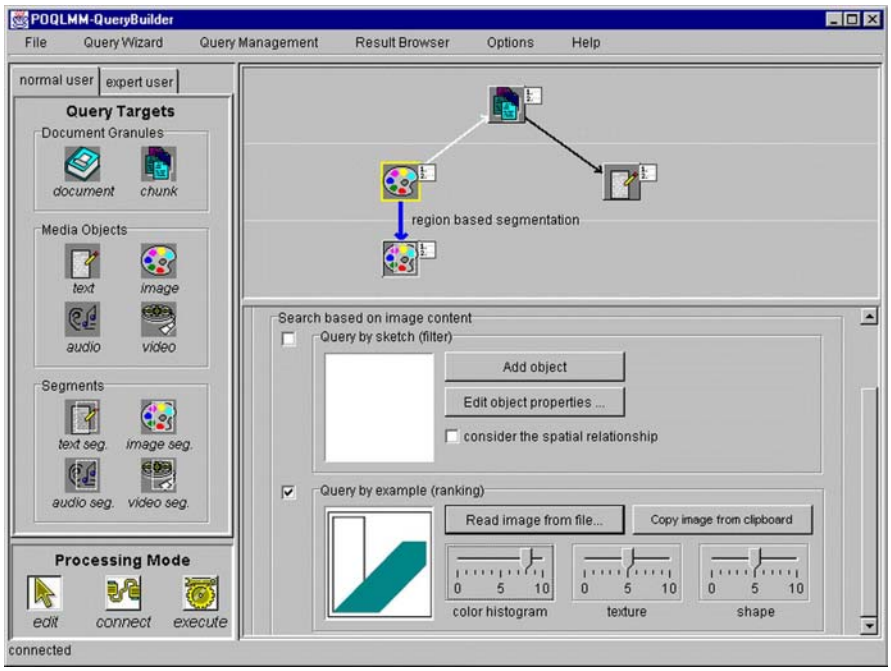


Abbildung 4: Beispiel einer Benutzungsoberfläche, die die Konzepte *Ranker*, *Combiner*, *Transferer* und *Filter* aufgreift

6 Überlegungen zur Benutzungsoberfläche

Aufbauend auf der prototypischen Implementierung unseres Systems (vgl. Abschnitt 7) haben wir eine Benutzungsoberfläche entwickelt, die *Ranker*, *Combiner*, *Transferer* und *Filter* zur Verfügung stellt und dem Benutzer die Möglichkeit gibt, seine Anfrage durch eine grafische Kombination dieser Komponenten zu definieren. Abbildung 4 zeigt diese Oberfläche, wobei exemplarisch die in Abschnitt 2 beschriebene Anfrage definiert wurde.

Zur Definition einer Anfrage können die Icons aus dem linken Teilfenster auf das Anfragedefinitionsfenster rechts oben gezogen werden. Zu den Icons, die im Wesentlichen die Objekttypen aus unserem Beispielschema repräsentieren, können dann Selektionsbedigungen und Ordnungskriterien definiert werden. In Abbildung 4 ist im unteren rechten Teil exemplarisch das entsprechende Formular für Bildsegmente dargestellt. Bei der Definition der Anfrage können auch die Semantiken für die einzelnen Transferoperationen definiert werden – hierzu existieren entsprechende Popup-Menus.

Bei der Definition zahlreicher Beispielanfragen hat sich die Oberfläche als leistungsfähig und intuitiv erwiesen. Sie macht deutlich, dass die arbeit mit Rankern, Combinern, Transferern, etc. nicht nur auf der Implementierungsebene sondern auch an der Endbenutzerschnittstelle sinnvoll sein kann. Die Details der Oberfläche sind in [HR01a] beschrieben.

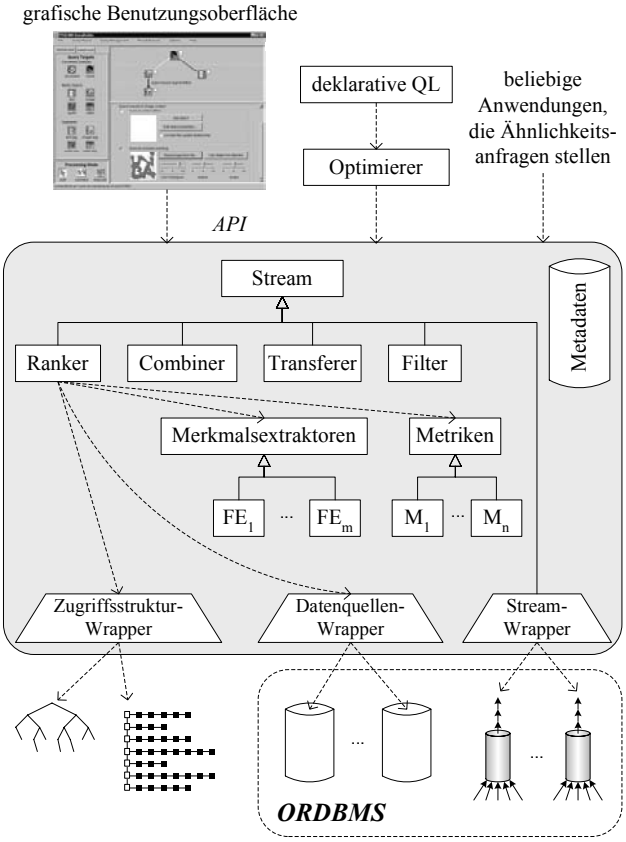


Abbildung 5: Architektur der prototypischen Implementierung

7 Systemarchitektur und experimentelle Ergebnisse

Die Architektur unserer prototypischen Implementierung basiert auf der Idee, die Daten in einem oder mehreren externen Datenspeichern abzulegen. Konkret verwenden wir hierzu eine objektrelationale Datenbank (ORDBMS). Die mit Strömen analog zu Pipes arbeitende *„retrieval engine“* ist in Java oberhalb des Datenspeichers realisiert und stellt eine Programmierschnittstelle (API) zur Verfügung, um die Realisierung ähnlichkeitsbasierter Suchdienste zu ermöglichen. Abbildung 5 verdeutlicht diese Architektur.

Der in Abbildung 5 grau hinterlegte Kern unseres Ansatzes besteht aus vier Hauptteilen: (1) Implementierungen für *Ranker*, *Combiner*, *Transferer* und *Filter*, (2) Implementierungen diverser Methoden für die Extraktion von Merkmalswerten (*feature values*) für verschiedene Objekttypen sowie entsprechender Ähnlichkeitsmaße, (3) einer Komponente zur Verwaltung von Metadaten für das System selbst und die Applikationen, die das System über die API nutzen und (4) Wrappern (oder *Konnektoren*), um externe Datenquellen,

Indexstrukturen und „Stromimplementierungen“ zu integrieren.

Ein **Merkmalsextraktor** erhält ein Objekt eines gegebenen Typs – z.B. ein Bild, ein Textobjekt oder eine Zeitreihe – und extrahiert hieraus einen Merkmalswert für dieses Objekt. Dieser Merkmalswert kann z.B. ein Farbhistogramm, ein Vektor zur Beschreibung der Textureigenschaften oder eine Vektorrepräsentation, die mit Hilfe des Vektorraummodells für einen Text erstellt wurde, sein. Die **Ähnlichkeitsmaße** erhalten zwei Merkmalsrepräsentationen als Eingabe – typischerweise eine, die ein Anfrageobjekt repräsentiert und eine, die ein Objekt aus einer Datenquelle repräsentiert – und bestimmen für dieses Paar einen Retrieval-Status-Wert.

Implementierungen eines *Rankers*, *Combiners*, *Transferers* oder *Filters* werden grundsätzlich von der Klasse **Stream** abgeleitet. Die Schnittstelle dieser Klasse besteht aus einem spezifischen Konstruktor und einer getNext-Methode.

Zum Beispiel erwartet der Konstruktor eines **Rankers** die Spezifikation der Datenquelle – in unserem System typischerweise eine Tabelle des ORDBMS –, einen Merkmalsextraktor, ein Ähnlichkeitsmaß und ein Anfrageobjekt. Der Konstruktor greift dann auf die Metadaten zu und ermittelt, ob für diese Konstellation aus Datenquelle, Merkmalsextraktor und Ähnlichkeitsmaß eine Indexstruktur existiert. In diesem Fall wird die Zugriffstruktur genutzt, um den Ranking-Prozess zu beschleunigen.

Für die Konstruktion eines **Combiners** werden zwei oder mehr Eingabeströme sowie eine entsprechende Gewichtung benötigt. Dabei ist beachtenswert, dass *Combiner* wie Fagin's Algorithmus oder Quick-Combine annehmen, dass auf den Objekten der Eingabeströme ein wahlfreier Zugriff unterstützt wird. Dazu muss der Produzent des Eingabestroms die Möglichkeit bieten, effizient den Retrieval-Status-Wert eines konkreten Objekts zu berechnen, das bisher noch nicht aus dem Eingabestrom gelesen wurde. Der Grund für diese Anforderung ist, das diese Algorithmen immer, wenn sie ein Objekt aus einem ihrer Eingabeströme lesen, versuchen, unmittelbar, den kombinierten Retrieval-Status-Wert für dieses Objekt zu berechnen. Hierzu führen sie einen wahlfreien Zugriff auf den anderen Eingabesrömen aus. Leider bieten einige Implementierungen für die Erzeugung eines Eingabestroms keinen wahlfreien Zugriff. In diesem Fall müssen andere Algorithmen zur Kombination der Eingabeströme verwendet werden – z.B. Nosferatu oder J^*.

Zur Konstruktion eines **Transferers** wird ein Eingabestrom, ein Pfadausdruck zur Definition der gewünschten Beziehung und eine Übertragungssemantik benötigt. Bei unserer Implementierung werden „references“ und „scoped references“, die das unterliegende ORDBMS anbietet, verwendet um die Pfadausdrücke zu definieren. Schließlich sind zur Konstruktion eines **Filters** ein Eingabestrom und eine Filterbedingung erforderlich.

Für jeden erzeugten Strom können nun die Elemente des Ausgabestroms schrittweise über die entsprechende getNext-Methode abgefragt werden.

In der **Metadaten**-Komponente unseres Systems werden Informationen zu den verfügbaren Merkmalsextraktoren, Ähnlichkeitsmaßen, Zugriffsstrukturen, etc. verwaltet. Diese Metadaten, die im System zur Anfrageoptimierung verwendet werden, können auch über die API zugegriffen werden. Hier können die Metadaten z.B. genutzt werden, um die Anfragekonstruktion in einer graphischen Benutzungsoberfläche zu steuern.

Wrapper für Datenspeicher werden benötigt, um Systeme, die die Objekte selbst verwalten, an unser Retrieval-System zu koppeln. Gegenwärtig existiert ein Wrapper zur Anbindung objektrelationaler Datenbanken über JDBC. Wrapper für Zugriffsstrukturen können genutzt werden, um Strukturen, die ursprünglich nicht für das System entwickelt wurden, zu integrieren. Zum Beispiel haben wir eine in C++ geschriebene Implementierung des LSD^h-Baumes über einen entsprechenden Wrapper angebunden. Schließlich können Wrapper für externe Stream-Implementierungen genutzt werden, um Komponenten zu integrieren, die Rangordnungen über Objekten erzeugen. Gegenwärtig wird das Textmodul des unterliegenden ORDBMS über einen solchen Wrapper eingebunden. Dabei stellt eine externe Stream-Implementierung nicht nur ein Ranking, sondern auch den Zugriff auf die verwalteten Objekte selbst zur Verfügung, während eine externe Zugriffsstruktur nur die Merkmalswerte und zugeordnete Objektreferenzen kennt.

Oberhalb der vom System zur Verfügung gestellten API können vielfältige Anwendungen realisiert werden. Ein Beispiel ist die graphische Benutzungsoberfläche aus Abschnitt 6. Ein anderes Beispiel ist die Implementierung einer deklarativen Anfragesprache auf Basis der API. Gegenwärtig arbeiten wir an einer entsprechenden Anpassung unserer Anfragesprache POQL$^{\mathrm{MM}}$ [Hen96, HR01b].

Um die Performance des vorgestellten Ansatzes zu überprüfen, haben wir die Bausteine, wie *Ranker*, *Combiner* und *Transferer*, in Java implementiert. Zur Verwaltung der Metadaten wird ein ORDBMS eingesetzt. Die hier vorgestellten Testergebnisse wurden mit Hilfe einer Dokumentenkollektion ermittelt, die Artikel einer Computerzeitschrift als strukturierte Dokumente enthält. Die Kollektion umfasst 2213 Artikel, 29414 Textblöcke, 4999 Bilder und 19980 Bildsegmente. Diese Dokumente wurden in das unserem System zugrunde liegende ORDBMS eingefügt. Ferner wurden zwei LSD^h-Bäume für Feature-Vektoren erzeugt, die die Farbcharakteristika und die Texturcharakteristika der Bildsegmente verwalten. Für die Farbähnlichkeit wurden zehndimensionale Farbhistogramme nach dem Munsell-Modell verwendet[SW95]. Für die Texturähnlichkeit wurden vierdimensionale Eigenschaftsvektoren genutzt, die die Homogenität, die Energie, den Kontrast und die Entropie der Bildsegmente beschreiben.

Wir haben drei unterschiedliche Arten von Anfragen auf diesen Testdaten ausgeführt: Anfragen, die nur einen *Ranker* nutzen, Anfragen, die zwei *Ranker* und einen darauf aufbauenden *Combiner* nutzen, und Anfragen, die zwei *Ranker*, einen *Combiner* und einen *Transferer* verwenden. Im letzten Fall wurde mit der Durchschnitts- und der Maximumsematik gearbeitet.

Mit der ersten Anfrage wurde nach den k im Hinblick auf die Farbähnlichkeit ähnlichsten Bildsegmenten, verglichen mit einem gegebenen Anfragebild, gesucht. Folglich wurde zur Bearbeitung dieser Anfrage genau ein *Ranker* benötigt. Abbildung 6a zeigt, dass die Performance unseres Systems in diesem Fall weit besser ist als wenn die Anfrage direkt mit den Möglichkeiten des ORDBMS ausgeführt wird. Dabei ist anzumerken, dass die für das ORDBMS angegebenen Werte die besten Werte sind, die sich nach dem Austesten verschiedenster SQL-Anfragen und Indexstrukturen auf der Datenbank ergaben.

In der zweiten Anfrage haben wir nach den k ähnlichsten Bildsegmenten verglichen mit einem Anfragebild gesucht, wobei nun sowohl die Farb- als auch die Texturähnlichkeit

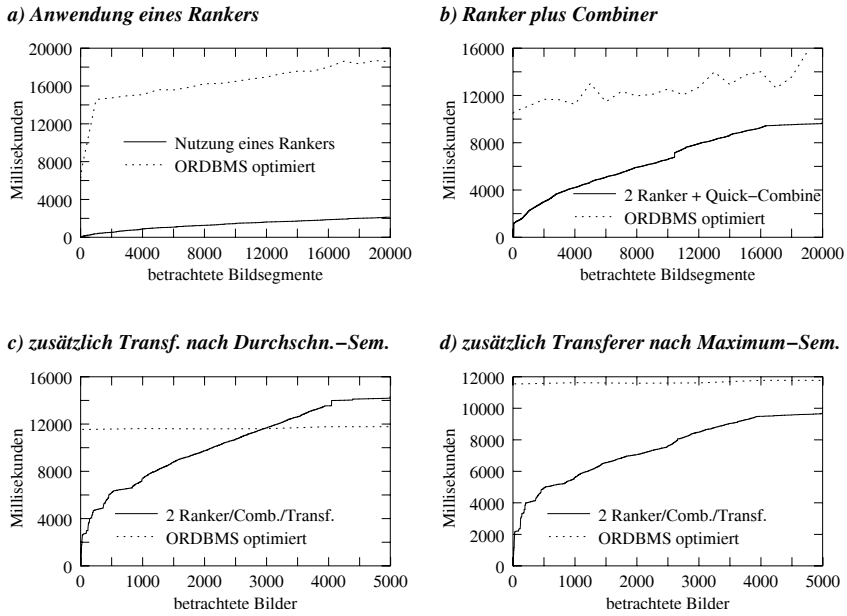


Abbildung 6: Messergebnisse für verschiedene Anfragen

betrachtet wurde. An der Ausführung dieser Anfrage in unserem Ansatz waren folglich zwei *Ranker* und ein *Combiner* (Quick-Combine) beteiligt. Abbildung 6b zeigt, dass auch bei dieser Anfrage unser Ansatz deutlich bessere Ergebnisse liefert als eine optimierte SQL-Anfrage auf dem Datenbanksystem. Auffällig sind die schwankenden Werte bei den Zeitmessungen für die Laufzeiten bzgl. des ORDBMS. Diese Schwankungen liegen darin begründet, dass sich die Messwerte für das ORDBMS durch die Mittelung der Laufzeitergebnisse von jeweils fünf Durchläufen an ausgewählten Messpunkten ergaben.

Schließlich wurde in der dritten Anfrage zusätzlich eine Übertragung des Rankings für die Bildsegmente auf die zugehörigen Bilder durchgeführt. Dazu wurde der in dieser Arbeit beschriebene *RSV-Transfer*-Algorithmus einmal mit der Durchschnittssemantik und einmal mit der Maximumsemantik angewendet. Die Abbildungen 6c und 6d zeigen die entsprechenden Ergebnisse. Es wird deutlich, dass selbst bei der für die Performance eher ungünstigen Durchschnittssemantik für realistische Werte von k eine deutliche Steigerung der Performance gegenüber dem ORDBMS erzielt werden konnte.

Bei der Interpretation der dargestellten Messergebnisse ist zu berücksichtigen, dass unsere Prototyp-Implementierung in Java sicher noch weiteres Optimierungspotential lässt. Trotzdem deuten schon diese Ergebnisse an, dass die Nutzung unseres Ansatzes mit expliziten Komponenten für die Übertragung eines Rankings eine konkurrenzfähige Performance erzielen kann.

8 Zusammenfassung und Ausblick

In der vorliegenden Arbeit haben wir einen expliziten Ansatz zur Übertragung einer Rangordnung, die für Objekte eines bestimmten Typs definiert ist, auf verbundene Objekte vorgestellt. Die explizite Betrachtung dieses Transfer-Schrittes im Rahmen einer komplexen Ähnlichkeitsanfrage hat den Vorteil, dass die Semantik der Übertragung und der entsprechende Algorithmus allgemeingültig betrachtet werden können. Eine auf diesem Ansatz basierende Benutzungsoberfläche macht deutlich, dass er auch zur graphischen Formulierung von Anfragen geeignet ist und eine prototypische Implementierung zeigt, dass die erreichbare Performance konkurrenzfähig ist.

Der in dieser Arbeit vorgeschlagene Ansatz optimiert dabei die Anfragebearbeitung bewusst im Hinblick auf Ranking-Bedingungen und die Nutzung entsprechender Zugriffsstrukturen und Algorithmen. Im Zusammenhang einer objektrelationalen Datenbank kann er deshalb auch im Sinne eines weiteren möglichen Ausführungsplans in den Optimierungsprozess mit einbezogen werden. Die Überlegungen hierzu sind Gegenstand unserer aktuellen Forschung. Dies gilt auch für Betrachtungen zur Ergebnisqualität, in deren Rahmen auf Basis von Benutzerstudien ermittelt werden soll, inwieweit strukturierte Ähnlichkeitsanfragen – wie die im Papier betrachtete Beispielanfrage – eine bessere Erfüllung der Informationswünsche der Benutzer erlauben als Ähnlichkeitsanfragen, die sich nur auf die Objekte selbst beziehen.

Literaturverzeichnis

[AC97] Anastasia Analyti and Stavros Christodoulakis. Content-Based Querying. In P.M.G. Apers, H.M. Blanken, and M.A.W. Houtsma, editors, *Multimedia Databases in Perspektive*, chapter 8, pages 145–180. Springer, Berlin, 1997.

[BCG02] Nicolas Bruno, Surajit Chaudhuri, and Luis Garvano. Top-*k* selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems*, 27(2):153–187, 2002.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree : An Index Structure for High-Dimensional Data. In *VLDB'96, Proc. 22th Intl. Conf. on Very Large Data Bases*, pages 28–39, Mumbai (Bombay), India, September 1996.

[BL85] Chris Buckley and A.F. Lewit. Optimization of inverted vector searches. In *Proc. of the 8th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 97–105, New York, USA, 1985.

[CK97] Michael J. Carey and Donald Kossmann. On Saying "Enough Already!" in SQL. In *Proc. of the 1997 ACM SIGMOD Intl. Conf. on Management of Data*, pages 219–230, Tucson, Arizona, 13–15 June 1997.

[CK98] Michael J. Carey and Donald Kossmann. Reducing the Braking Distance of an SQL Query Engine. In *VLDB'98, Proc. 24th Intl. Conf. on Very Large Data Bases*, pages 158–169, New York, USA, 1998.

[Fag98] Ronald Fagin. Fuzzy Queries in Multimedia Database Systems. In *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–10, Seattle, Washington, 1998. ACM Press.

[FLN01] Ronald Fagin, Ammon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. In *Proc. of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Santa Barbara, California, USA, May 2001.

[GBK00] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing Multi-Feature Queries for Image Databases. In *VLDB 2000, Proc. 26th Intl. Conf. on Very Large Data Bases*, pages 419–428, Cairo, Egypt, September 2000.

[GBK01] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Towards Efficient Multi-Feature Queries in Heterogeneous Environments. In *Intl. Conf. on Information Technology: Coding and Computing (ITCC 2001)*, pages 622–628, Las Vegas, USA, 2001.

[Hen96] Andreas Henrich. Document Retrieval Facilities for Repository-Based System Development Environments. In *Proc. of the 19th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 101–109, Zürich, 1996.

[Hen98] Andreas Henrich. The LSD$^{\mathrm{h}}$-Tree: An Access Structure for Feature Vectors. In *Proc. of the 14th Intl. Conf. on Data Engineering, February, 1998, Orlando, Florida*, pages 362–369. IEEE Computer Society, 1998.

[HR01a] Andreas Henrich and Günter Robbert. An End User Retrieval Interface for Structured Multimedia Documents. In *Proc. 7th Workshop on Multimedia Information Systems, MIS'01*, pages 71–80, Capri, Italy, 2001.

[HR01b] Andreas Henrich and Günter Robbert. POQL$^{\mathrm{MM}}$: A Query Language for Structured Multimedia Documents. In *Proc. 1st Intl. Workshop on Multimedia Data and Document Engineering, MDDE'01*, pages 17–26, Lyon, France, 2001.

[IAE02] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Joining Ranked Inputs in Practice. In *VLDB'02, Proc. of 28th Intl. Conf. on Very Large Data Bases*, Hong Kong, China, 2002.

[NCS+01] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting Incremental Join Queries on Ranked Inputs. In *VLDB 2001, Proc. of 27th Intl. Conf. on Very Large Data Bases*, pages 281–290, Roma, Italy, 9 2001.

[NR99] Surya Nepal and M. V. Ramakrishna. Query Processing Issues in Image (Multimedia) Databases. In *Proc. of the 15th Intl. Conf. on Data Engineering*, pages 22–29, Sydney, Austrialia, 1999. IEEE Computer Society.

[PP97] Ulrich Pfeifer and Stefan Pennekamp. Incremental Processing of Vague Queries in Interactive Retrieval Systems. In *HIM '97, Proc. Hypertext - Information Retrieval - Multimedia '97*, pages 223–235, Dortmund, 1997. Universitätsverlag Konstanz.

[Sal89] G. Salton. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, Mass., 1989.

[SMMR99] D. M. Squire, W. Müller, H. Müller, and J. Raki. Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback. In *The 11th Scandinavian Conf. on Image Analysis (SCIA 99)*, pages 143–149, Kangerlussuaq, Greenland, 1999.

[SW95] J. Sturges and T.W.A. Whitfield. Locating basic colours in the munsell space. *Color Research and Application*, 20:364–376, 1995.

[VGJL94] E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird. The Collection Fusion Problem. In *Proc. of the 3rd Text Retrieval Conf. (TREC-3)*, pages 95–104, Gaithersburg, Maryland, 11 1994. NIST Special Publication 500 - 225.

[WSB98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB'98, Proc. of 24rd Intl. Conf. on Very Large Data Bases*, pages 194–205, New York, 1998.

[ZSAR98] Pavel Zezula, Pasquale Savino, Giuseppe Amato, and Fausto Rabitti. Approximate Similarity Retrieval with M-Trees. *VLDB Journal*, 7(4):275–293, 1998.

Relevanzgewichtung in komplexen Ähnlichkeitsanfragen

Nadine Schulz Ingo Schmitt

{nschulz|schmitt}@iti.cs.uni-magdeburg.de

Zusammenfassung: Eine Möglichkeit, um Nutzerpräferenzen in heutigen Multimedia- und Information Retrieval Systemen abzubilden, besteht in der Gewichtung von Anfragetermen. Die meisten Arbeiten auf diesem Gebiet sind in ihrer Anwendung auf das Gewichten einfacher Anfragen beschränkt. Aus diesem Grund wird in diesem Papier die Relevanzgewichtung auf verschiedenen Ebenen von komplexen Anfragen betrachtet. Es werden zwei Alternativen zur Formulierung von Gewichten in komplexen Anfragen demonstriert. Des Weiteren werden Transformationsregeln für gewichtete Anfragen untersucht, die eine Optimierung der Anfragebearbeitung ermöglichen.

1 Einführung und Motivation

In den vergangenen 20 Jahren entwickelte sich das Gewichten von Anfragetermen als eine Möglichkeit, um Nutzerpräferenzen in Multimedia und Information Retrieval Systemen abzubilden [DP86, FW97, SFW83, Sun98, WK79, Yag87]. Die Vergabe von Relevanzgewichten für bestimmte Anfrageterme in Fuzzy oder Multimedia-Anfragen ermöglicht dem Nutzer eine flexible Spezifikation von Präferenzen. Für einfache Anfragen, also Anfragen, die jeweils nur aus der Konjunktion bzw. Disjunktion von Anfragetermen bestehen, können derzeitige Verfahren [Yag87, DP86, FW97, Sun98] zur Handhabung von Relevanzgewichten erfolgreich eingesetzt werden, obwohl sie verschiedene Schwachstellen, wie in [Sun98, FW00] diskutiert, aufweisen. Bei diesem Typ von Anfragen wird die Gewichtung eines Anfrageterms jeweils gegenüber allen anderen Termen in der Anfrage betrachtet.

Im Gegensatz zu den einfachen Anfragen stehen die komplexen Anfragen. Hier können einzelne Terme oder Teilanfragen gegenüber anderen Teilanfragen gewichtet werden. Damit ist in komplexen Anfragen eine Gewichtung auf verschiedenen Ebenen möglich. Zwei Arten von komplexen Anfragen können unterschieden werden. Heterogene komplexe Anfragen bestehen sowohl aus Konjunktionen als auch aus Disjunktionen von Anfragetermen. Homogene komplexe Anfragen hingegen bestehen jeweils nur aus der Konjunktion bzw. Disjunktion von Anfragetermen.

Derzeitig gibt es keine Lösung für die Verarbeitung von Gewichten auf verschiedenen Ebenen in komplexen Anfragen. Ziel ist es daher, in komplexen Anfragen eine Gewichtung auf allen Ebenen zu ermöglichen und Transformationsregeln für die Optimierung dieser Anfragen bereitzustellen. Es wird die Entwicklung eines neuen Ansatzes, der als Multi-

Level-Gewichten bezeichnet wird, angestrebt.

Ein weiterer Aspekt, der bei der Entwicklung unseres Ansatzes betrachtet werden soll, ist die syntaktische Umformung von Anfragen in andere, für die Optimierung besser geeignete Anfrageformen durch das Retrieval System. Die syntaktische Transformation von Anfragen in logisch äquivalente Anfragen ist eine wichtig Eigenschaft von Retrieval Systemen mit Booleschen Operationen. Während die Transformation von ungewichteten Booleschen Anfragen ohne Probleme möglich ist, stellt die Transformation von gewichteten Anfragen ein bisher offenes Problem in Retrieval Systemen dar [Boo78, WK79]. Bei der Transformation muss sichergestellt werden, dass für syntaktisch unterschiedliche jedoch semantisch äquivalente, gewichtete Anfragen stets das gleiche Anfrageergebnis berechnet wird. Zur Umgehung dieses Problems werden die Nutzer derzeitig oftmals bei der Formulierung ihrer Anfragen eingeschränkt, indem sie beispielsweise ihre Anfrage direkt in disjunktiver Normalform (DNF) bzw. konjunktiver Normalform (KNF) formulieren müssen [HV01, Pas99]. Diese Herangehensweise stellt für den Nutzer eine zu starke Einschränkung dar. Aus diesem Grund bildet die Entwicklung von geeigneten logischen Transformationsregeln für das Multi-Level-Gewichtungsmodell einen Schwerpunkt in diesem Papier.

Im Weiteren ist die Arbeit wie folgt gegliedert. In Abschnitt 2 wird das zugrundeliegende Anfragemodell sowie, die in diesem Papier verwendeten Notationen dargelegt. Abschnitt 3 beschreibt zwei Alternativen zur Formulierung von gewichteten Anfragen. Weiterhin werden in Abschnitt 4 Anforderungen der logischen Umformung von gewichteten Anfragen erläutert. Es wird auf entsprechende Umformungsregeln für gewichtete, komplexe Anfragen eingegangen. Die Evaluierung dieser Anfragen erfolgt durch gewichtete Scoring-Funktionen, die mit dem Ansatz von Fagin und Wimmers [FW97] generiert werden. Abschließend wird ein Überblick über offenen Probleme und weiteren Forschungsaufgaben gegeben.

2 Anfragemodell

Eine Anfrage X besteht aus n atomaren Suchbedingungen x_i, die durch die Junktoren *und* ($\wedge$) und *oder* ($\vee$) miteinander verknüpft sind. Weiterhin, besteht die Möglichkeit Anfrageterme zu negieren:

$$X := x \mid (X\ [\wedge|\vee]\ X) \mid \neg X.$$

Eine komplexe Anfrage, wie in Abbildung 1 dargestellt, umfasst verschiedene Ebenen l_j mit $j = 0, \ldots, m$, so dass $m + 1$ die Gesamtanzahl der Ebenen angibt. Jede Ebene l_j enthält n_j verschiedene Teilanfragen $s_{j,i}$ mit $i = 1, \ldots, n_j$, wobei jede *und/oder*-Teilanfrage aus zwei Operanden besteht. Zur Bestimmung der direkten Kinder einer Teilanfrage $s_{j,i}$ wird die Funktion $child(s_{j,i})$ verwendet.

Nutzerpräferenzen können durch nummerische Gewichte θ_i mit $\theta_i \in [0, 1]$ und $\sum_{i=1}^{n} \theta_i = 1$ ausgedrückt werden. Typischerweise wird jedes Gewicht θ_i jeweils dem atomaren Anfrageterm x_i zugeordnet. Die Funktion $weight()$ ermittelt das Gewicht eines atomaren Anfrageterms oder einer Teilanfrage.

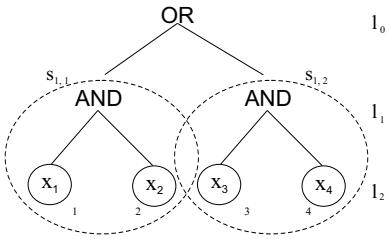


Abbildung 1: Anfragebaum einer komplexen Anfrage mit den Ebenen l_j, Teilanfragen $s_{j,i}$ und Gewichten θ_i

Bei der Evaluierung einer Anfrage X wird für jedes Objekt in der Datenbank und jede atomare Suchbedingung x_i ein Score μ_i im Intervall $[0, 1]$ berechnet, der ausdrückt inwieweit sich das Objekt die Suchbedingung x_i erfüllt. Je höher der Score μ_i ist, desto höher ist der Grad der Ähnlichkeit zwischen den Objekten bezüglich der Suchbedingung x_i. Anschließend wird für jedes Objekt der Gesamtscore μ berechnet. Dafür werden mittels der Junktoren *und* und *oder* jeweils zwei Fuzzy Mengen kombiniert und letztendlich eine neue Fuzzy-Menge als Gesamtscore für jedes Objekt in der Datenbank ermittelt. Die Kombination der Fuzzy Mengen für ein Objekt wird mittels einer Scoring-Funktion

$$S_X : [0, 1]^n \rightarrow [0, 1]$$

für eine Anfrage X realisiert. Typischerweise basiert S_X auf T-Normen und T-Conormen. Eine Konjunktion von Anfragetermen wird evaluiert unter Verwendung einer T-Norm und eine Disjunktion von Anfragetermen dementsprechend unter Verwendung einer T-Conorm (siehe [Zad65]). Der Einsatz von Fuzzy-basierten Scoring-Funktionen ist weit verbreitet, wie z. B. in den Ansätzen von [HV01, ORC+98, CBGM97]. Andere Scoring-Funktionen, wie beispielsweise probabilistische Funktionen, finden ebenso Anwendung [FG01].

Im Falle einer gewichteten Anfrage, wobei jeder Anfrageterm x_i mit einem Gewicht θ_i versehen ist, müssen die Gewichte der Anfrageterme mit in die Scoring-Funktion S_X einfließen. Es ergibt sich die gewichtete Scoring-Funktion

$$S_X^{\Theta} : [0, 1]^n \times [0, 1]^n \rightarrow [0, 1].$$

3 Multi-Level Gewichtung

Typischerweise wird jedes Gewicht einem atomaren Anfrageterm zugeordnet. Diese Einschränkung der Gewichtung auf atomare Terme ist in komplexen Anfragen nicht praktikabel, da dieses nicht dem Denkmuster des Nutzers entspricht und dieser daher die Auswirkungen seiner Gewichtung oft nicht nachvollziehen kann. Vielmehr sollte der Nutzer die Freiheit haben, Gewichte auf allen Ebenen einer komplexen Anfrage zu spezifizieren.

3.1 Implizites versus Explizites Gewichten

Für die Gewichtung von komplexen Anfragen adaptieren wir den Ansatz von [SSS02]. Demnach werden folgende zwei Alternativen unterschieden:

1. Implizites Gewichten: der Nutzer ordnet, wie auch bei einfachen Anfragen, jedem atomaren Term x_i ein Gewicht θ_i zu. Dabei gilt als globale Beschränkung $\sum_{i=1}^{n} \theta_i = 1$. Unter Berücksichtigung der gegebenen Gewichte für die atomaren Terme und der Semantik der Anfrage sind die Gewichte für die Teilanfragen $s_{j,i}$ auf einer höheren Ebene damit implizit bestimmt. Sie können aus den zugrundeliegenden atomaren Termgewichten ermittelt werden. Das Gewicht für eine Teilanfrage $s_{j,i}$ wird mittels einer Funktion f aus den Gewichten der Kinder der Teilanfrage berechnet. Einer Teilanfrage $s_{j,i}$ wird das Gewicht $\theta_{j,i} = f(weight(child(s_{j,i}))$ zugeordnet. Die implizite Gewichtung wird mit Θ bezeichnet.

2. Explizites Gewichten: der Nutzer spezifiziert explizit ein Gewicht θ_i für jeden atomaren Term x_i und ein Gewicht $\theta_{j,i}$ für jede Teilanfrage $s_{j,i}$. Die explizite Gewichtung wird mit Θ^E bezeichnet. Im Gegensatz zum impliziten Gewichten gibt es lokale Beschränkungen. Für jede Teilanfrage $s_{j,i}$ gilt, dass die Gewichte ihrer Kinder sich zu 1 summieren: $\forall s_{j,i} : \sum_{\theta_{j,t} \in weight(child(s_{j,i}))} \theta_{j,t} = 1$. Auf Grund der lokalen Beschränkung entspricht das Gewicht für die Teilanfrage $s_{0,1}$ immer 1, so dass das Gewicht $\theta_{0,1}$ nicht mit in die Gewichtung Θ^E aufgenommen wird.

Beide Alternativen der Formulierung von Gewichten unterscheiden sich in der Art der Spezifikation der Gewichte sowie in der Anzahl der zugrundeliegenden Beschränkungen. Jedoch verfügen sie über die gleiche Mächtigkeit. Die Gewichtung der Anfrageterme und Teilanfragen kann vom Anwender ohne Berücksichtigung der Bedingungen bei der Anfrageformulierung vorgenommen werden. Intern erfolgt eine Normalisierung der Gewichte, so dass die lokalen bzw. globalen Beschränkungen vom System eingehalten werden.

Als Initialgewichtung für eine Anfrage wird angenommen, dass alle Gewichte auf den einzelnen Ebenen gleich sind, was einer ungewichteten Anfrage entspricht. Dies bietet dem Nutzer die Möglichkeit, Gewichte nur für bestimmte Anfrageterme anzugeben. Eine andere Möglichkeit für die Festlegung der Initialgewichte besteht in der Verwendung von Nutzerprofilen.

In diesem Ansatz werden nummerische Gewichte verwendet. Jedoch ist es ebenso denkbar, statt dessen Fuzzy-Gewichte zu verwenden, die durch linguistische Variablen, wie z. B. *sehr wichtig*, *unwichtig*, usw. ausgedrückt werden können [HV01].

Im Folgenden zeigen wir die Evaluierung einer komplexen Anfrage mittels einer gewichteten Scoring-Funktion. Hierfür muss die Gewichtung in expliziter Form vorliegen. Daher ist es gegebenenfalls notwendig, eine implizite Gewichtung in die entsprechende explizite Gewichtung zu überführen. Für diese Umformung der Gewichtungen verwenden wir den in [SSS02] vorgestellten Ansatz. Die Überführung der beiden Gewichtungsformen ineinander ist invers und stellt somit eine Bijektion dar.

3.2 Gewichtete Multi-Level Scoring-Funktionen S_X^Θ

Für die Evaluierung von komplexen, gewichteten Anfragen sind Scoring-Funktionen vonnöten, die eine Handhabung von Gewichten auf den verschiedenen Ebenen ermöglichen. Hierfür wird auf gewichtete Scoring-Funktionen für einfache Anfragen zurückgegriffen [Yag87, DP86, FW97, Sun98], welche bereits erfolgreich eingesetzt werden. Da in komplexen Anfragen jede *und/oder*-Teilanfrage $s_{j,i}$ einer einfachen Anfrage entspricht kann durch die rekursive Anwendung der Scoring-Funktionen, die ursprünglich für einfache Anfragen entwickelt wurden, eine Gewichtung auf mehreren Ebenen einer komplexen Anfrage gewährleistet werden.

Wir greifen hier auf den Ansatz von Fagin und Wimmers zurück [FW97]. Dieser erlaubt die Gewichtung einer beliebigen zugrundeliegenden Scoring-Funktion S_X mit $\Theta = \langle\theta_1, \ldots, \theta_n\rangle$, wobei jedes Gewicht θ_i einem Anfrageterm x_i zugeordnet ist. Unter Beachtung der Annahmen $\theta_i \in [0,1]$, $\sum_{i=1}^{n} \theta_i = 1$ und $\theta_1 \geq \theta_2 \geq \ldots \geq \theta_n$, welche mittels der Kommutativität erreicht wird, kann eine gewichtete Scoring-Funktion für einfache, gewichtete Anfragen mit folgender Formel generiert werden:

$$\begin{aligned} S_X^\Theta(\mu_1, \ldots, \mu_n) \quad = \quad & (\theta_1 - \theta_2) S_X(\mu_1) + 2 * (\theta_2 - \theta_3) S_X(\mu_1, \mu_2) \ldots + \ldots \\ & n * \theta_n S_X(\mu_1, \ldots, \mu_n) \end{aligned} \tag{1}$$

Eine gewichtete Scoring-Funktion für komplexe Anfragen kann mit Hilfe der Formel (1) rekursiv gebildet werden. Scoring-Funktionen für komplexe Anfragen werden als Multi-Level Scoring-Funktionen bezeichnet. Für die Generierung einer Multi-Level Scoring-Funktion ist es notwendig, dass die Gewichtung für komplexe Anfragen in expliziter Form vorliegt bzw. in diese überführt wird.

4 Transformation von Anfragen

Eine Fähigkeit von Retrieval Systemen mit Booleschen Operationen ist die syntaktische Transformation von Anfragen in spezielle Anfrageformen. Durch die syntaktische Veränderung der internen Darstellung der Anfrage ist eine Minimierung des Ressourcenverbrauchs und damit eine performante Anfragebearbeitung möglich. Beispielsweise kann eine Anfrage syntaktisch so umgeformt werden, dass der Score für eine Teilanfrage effizient in einem System und der Score für eine andere Teilanfrage in einem anderen System ermittelt werden kann. Eine syntaktische Transformation von Anfragen setzt voraus, dass die Evaluierung äquivalenter Anfrageformen immer das selbe Ergebnis liefert.

4.1 Transformation ungewichteter Anfragen

Grundlage für die Transformation von Anfragen bilden logische Transformationsregeln. Für eine Boolesche Formel der Aussagen- bzw. der Fuzzy-Logik, müssen die allgemei-

nen Transformationsregeln, wie Kommutativität, Assoziativität, Distributivität, De Morgan, Idempotenz und Involution, gelten [Boo78].

Die Transformation von ungewichteten Anfragen beruht auf diesen Transformationsregeln und ist ohne Probleme möglich. Demgegenüber stellt die Transformation von gewichteten Anfragen jedoch ein bisher offenes Problem dar, da bei der Anfrageevaluierung der Einfluss der Gewichte berücksichtigt werden muss. Aus diesem Grund werden Nutzer in verschiedenen Systemen gezwungen, ihre gewichteten Anfragen in der DNF bzw. CNF zu formulieren [HV01, Pas99]. Diese Herangehensweise ist für den Nutzer nicht praktikabel. Vielmehr sollte der Nutzer die Freiheit besitzen, seinen Informationsbedarf in einer beliebigen Anfrageform auszudrücken. Die Last der logischen Umformung sollte daher nicht dem Nutzer, sondern dem Retrieval System im Rahmen der Anfrageoptimierung auferlegt werden.

Im folgenden Abschnitt beschreiben wir eine Möglichkeit der Transformation von gewichteten Anfragen. Unser Ansatz kann gewährleisten, dass für syntaktisch unterschiedliche Anfrageformen einer gewichteten Anfragen gleiche Ergebnisse ermittelt werden.

4.2 Transformation von gewichteten Anfragen

Für die Evaluierung gewichteter Anfragen verwenden wir gewichtete Scoring-Funktionen, die mittels der Faginschen Formel aus Abschnitt 3.2 generierten werden. Dafür werden hier konkret die Scoring-Funktionen MIN für Konjunktionen und MAX für Disjunktionen von Anfragetermen verwendet. Bei Fagin und Wimmers wird ein Gewicht jeweils einem Anfrageterm zugeordnet, so dass sich ein Paar (x_i, θ_i) mit dem Term x_i und dem Gewicht θ_i ergibt. Bei komplexen Anfragen gehen wir davon aus, dass die Gewichtung in expliziter Form vorliegt.

Für gewichtete, einfache Anfragen, die mit einer Faginschen Scoring-Funktion evaluiert werden, gelten die Eigenschaften Kommutativität, Involution, De Morgan und Idempotenz:

- Kommutativität: $((x_1, \theta_1) \wedge (x_2, \theta_2)) = ((x_2, \theta_2) \wedge (x_1, \theta_1))$,
  $((x_1, \theta_1) \vee (x_2, \theta_2)) = ((x_2, \theta_2) \vee (x_1, \theta_1))$.

  Bei der Berechnung des Gesamtscores mittels einer gewichteten Scoring-Funktion werden unter Ausnutzung der Kommutativität die Anfragen so umgeformt, dass die Bedingung der Faginschen Formel $\theta_1 \geq \ldots \geq \theta_n$ gilt.

- Idempotenz: Wenn $x_1 = x_2$, dann gilt: $((x_1, \theta_1) \wedge (x_2, \theta_2)) = x_1 = x_2$,
  $((x_1, \theta_1) \vee (x_2, \theta_2)) = x_1 = x_2$.

- Involution: $\neg\neg(x_1, \theta_1) = (x_1, \theta_1)$.

- De Morgan: $\neg((x_1, \theta_1) \wedge (x_2, \theta_2)) = ((\neg x_1, \theta_1) \vee (\neg x_2, \theta_2))$,
  $\neg((x_1, \theta_1) \vee (x_2, \theta_2)) = ((\neg x_1, \theta_1) \wedge (\neg x_2, \theta_2))$.

Die Gültigkeit dieser Eigenschaften kann sehr einfach bewiesen werden. Aus Platzgründen haben wir die Beweise an dieser Stelle nicht mit angegeben und verweisen auf [SSnt]. Bei einfachen, gewichteten Anfragen sind Distributivität und Assoziativität nicht von Bedeutung, jedoch bei komplexen, gewichteten Anfragen. Um bei diesem Typ von Anfragen die Distributivität und Assoziativität zu gewährleisten ist es notwendig, Gewichte zu modifizieren. Im Folgenden gehen wir auf diese beiden Transformationsregeln näher ein.

Distributivität

Es gilt:

$$\begin{aligned}((((x_1,\theta_1)\wedge(x_2,\theta_2)),\theta_{1,1})\vee(x_3,\theta_3)) &= ((((x_1,\theta_{1'})\vee(x_3,\theta_{3'})),\theta_{1,1'})\\ &\quad\wedge(((x_2,\theta_{2'})\vee(x_3,\theta_{3''})),\theta_{1,2'})),\\ ((((x_1,\theta_1)\vee(x_2,\theta_2),\theta_{1,1}))\wedge(x_3,\theta_3)) &= ((((x_1,\theta_{1'})\wedge(x_3,\theta_{3'}),\theta_{1,1'})\\ &\quad\vee((x_2,\theta_{2'})\wedge(x_3,\theta_{3''})),\theta_{1,2'})).\end{aligned}$$

In unseren Betrachtungen unterscheiden wir zwischen einer *und/oder*-Anfrage ($X_{\wedge,\vee}$) sowie einer *oder/und*-Anfrage ($X_{\vee,\wedge}$). Wichtig bei der logischen Transformation von Anfragen ist, dass sowohl für die ursprüngliche Anfrage X als auch für die transformierte Anfrage X' gleiche Ergebnisse ermittelt werden. Aus diesem Grund wird bei der distributiven Transformation einer komplexen Anfrage die Modifikation der Gewichte erforderlich. Wir gehen davon aus, dass $\theta_1 \geq \theta_2$ und $\theta_{1,1} \geq \theta_3$ gilt. Die Gewichte für die transformierte Anfrage können dann wie folgt aus den Gewichten der ursprünglichen Anfrage berechnet werden:

$$\theta_{3'} = \theta_3$$

$$\theta_{1,2'} = \begin{cases} \frac{1-(1-2\theta_3)(1-2\theta_2)}{2} & X_{\wedge,\vee}:\mu_1\geq\mu_3\geq\mu_2 \text{ oder } X_{\vee,\wedge}:\mu_2\geq\mu_3\geq\mu_1 \\ \theta_2 & \text{sonst} \end{cases}$$

$$\theta_{3''} = \begin{cases} \frac{\theta_3}{1-(1-2\theta_2)(1-2\theta_2)} & X_{\wedge,\vee}:\mu_1\geq\mu_3\geq\mu_2\wedge \\ & \quad(1-2\theta_2)\mu_1+2\theta_2\mu_2<\mu_3 \text{ oder} \\ & X_{\vee,\wedge}:\mu_1\geq\mu_3\geq\mu_2\wedge \\ & \quad(1-2\theta_2)\mu_1+2\theta_2\mu_2>\mu_3 \\ 0 & X_{\wedge,\vee}:\mu_1\geq\mu_3\geq\mu_2\wedge \\ & \quad(1-2\theta_2)\mu_1+2\theta_2\mu_2\geq\mu_3 \text{ oder} \\ & X_{\vee,\wedge}:\mu_1\geq\mu_3\geq\mu_2\wedge \\ & \quad(1-2\theta_2)\mu_1+2\theta_2\mu_2\leq\mu_3 \\ \theta_3 & \text{sonst} \end{cases}$$

Auf Grund der lokalen Beschränkungen können die Gewichte $\theta_{1'}, \theta_{2'}$ sowie $\theta_{1,1'}$ aus den bereits ermittelten Gewichten berechnet werden, z. B. $\theta_{1'} = 1 - \theta_{3'}$.

Der folgende Beweis für den Fall $\mu_1 \geq \mu_2 \geq \mu_3$ und einer *oder/und*-Anfrage zeigt die Korrektheit unserer ermittelten Gewichte. Die Beweise für die anderen Fälle sind analog. Unsere Ergebnisse haben wir ebenfalls experimentell bestätigt.

Beweis: Es soll gelten $S_X^{\Theta^E} = S_{X'}^{\Theta^E}$. Unter Verwendung der Faginschen Formel ergibt sich für X:

$$\begin{aligned} S_{s_{1,1}}^{\Theta} &= (1-2\theta_2)\mu_1 + 2\theta_2 MAX(\mu_1,\mu_2) = \mu_1 \\ S_X^{\Theta^E} &= (1-2\theta_3)S_{s_{1,1}}^{\Theta} + 2\theta_3 MIN(S_{s_{1,1}}^{\Theta},\mu_3) \\ &= (1-2\theta_3)\mu_1 + 2\theta_3\mu_3 \end{aligned}$$

und für X':

$$\begin{aligned} S_{s_{1,1}}^{\Theta} &= (1-2\theta_{3'})\mu_1 + 2\theta_{3'} MIN(\mu_1,\mu_3) \\ S_{s_{1,2}}^{\Theta} &= (1-2\theta_{3''})\mu_2 + 2\theta_{3''} MIN(\mu_2,\mu_3) \\ S_{X'}^{\Theta^E} &= (1-2\theta_{1,2'})S_{s_{1,1}}^{\Theta} + 2\theta_{1,2'} MAX(S_{s_{1,1}}^{\Theta}, S_{s_{1,2}}^{\Theta}) \\ &\quad \text{mit } \theta_{3'} = \theta_3, \theta_{3''} = \theta_3, \theta_{1,2'} = \theta_2 \\ &= S_{s_{1,1}}^{\Theta} \\ &= (1-2\theta_3)\mu_1 + 2\theta_3\mu_3 = S_X^{\Theta^E} \qquad q.e.d. \end{aligned}$$ □

Assoziativität

Es gilt:
$((((x_1,\theta_1) \wedge (x_2,\theta_2)),\theta_{1,1}) \wedge (x_3,\theta_3)) = ((x_1,\theta_{1'}) \wedge (((x_2,\theta_{2'}) \wedge (x_3,\theta_{3'})),\theta_{1,1'}))$,
$((((x_1,\theta_1) \vee (x_2,\theta_2)),\theta_{1,1}) \vee (x_3,\theta_3)) = ((x_1,\theta_{1'}) \vee (((x_2,\theta_{2'}) \vee (x_3,\theta_{3'})),\theta_{1,1'}))$

Die Herangehensweise zur Bestimmung der Gewichte für die transformierte Anfrage ist hier analog zur distributiven Transformation. Die Gewichte für die transformierte Anfrage können wie im Folgenden angegeben berechnet werden. Die übrigen Gewichte, also $\theta_{1'}$ und $\theta_{2'}$, werden auf Grund der lokalen Beschränkungen aus den bereits ermittelten Gewichten berechnet.

$$\theta_{1,1'} = \begin{cases} \theta_3 & X_\wedge : \mu_2 \geq \mu_1 \geq \mu_3 \text{ oder } X_\vee : \mu_3 \geq \mu_1 \geq \mu_2 \\ \frac{1-(1-2\theta_3)(1-2\theta_2)}{2} & X_\wedge : \mu_1 \geq \mu_2 \geq \mu_3, \mu_1 \geq \mu_3 \geq \mu_2 \wedge \\ & \quad (1-2\theta_2)\mu_1 + 2\theta_2\mu_2 > \mu_3 \text{ oder} \\ & X_\vee : \mu_3 \geq \mu_2 \geq \mu_1, \mu_2 \geq \mu_3 \geq \mu_1 \wedge \\ & \quad (1-2\theta_2)\mu_1 + 2\theta_2\mu_2 < \mu_3 \\ \theta_2 & \text{sonst} \end{cases}$$

$$\theta_{3'} = \begin{cases} 1 & X_\wedge : \mu_2 \geq \mu_1 \geq \mu_3 \text{ oder } X_\vee : \mu_3 \geq \mu_1 \geq \mu_2 \\ \frac{\theta_3}{2\theta_{1,1'}} & X_\wedge : \mu_1 \geq \mu_2 \geq \mu_3 \text{ oder } X_\vee : \mu_3 \geq \mu_2 \geq \mu_1 \\ 1 - \frac{(1-2\theta_3)\theta_2}{2\theta_{1,1'}} & X_\wedge : \mu_1 \geq \mu_3 \geq \mu_2 \wedge (1-2\theta_2)\mu_1 + 2\theta_2\mu_2 > \mu_3 \\ & \text{oder} \\ & X_\vee : \mu_2 \geq \mu_3 \geq \mu_1 \wedge (1-2\theta_2)\mu_1 + 2\theta_2\mu_2 < \mu_3 \\ \theta_2 & \text{sonst} \end{cases}$$

Für die weitere Vereinfachung von gewichteten Anfragen geben Fagin und Wimmers zwei Regeln unter Berücksichtigung des Einflusses der Gewichte an [FW97]:

- Wenn ein atomarer Anfrageterm oder eine Teilanfrage mit Null gewichtet ist, dann kann dieser Term bzw. diese Teilanfrage von der Anfrage entfernt werden. Das Entfernen hat keinen Einfluss auf das Anfrageergebnis.
- Sind alle Gewichte gleich, also $\theta_i = 1/n$, entspricht die gewichtete Anfrage der ungewichteten Anfrage. Die Evaluierung der Anfrage kann ohne Berücksichtigung der Gewichte erfolgen, da in diesem Fall der gleiche Ergebniswert wie für die gewichteten Anfrage berechnet wird.

5 Zusammenfassung und Ausblick

Es wurde eine Methode zur Spezifikation von Relevanzgewichten für Anfrageterme in komplexen Anfragen beschrieben. Dieser Ansatz wird als Multi-Level-Gewichten bezeichnet, wobei zwei Alternativen der Gewichtung unterschieden werden. Bei der impliziten Gewichtung wird jedem atomaren Anfrageterm ein Gewicht zugeordnet. Gewichte für die Teilanfragen in komplexen Anfragen werden dann aus den gegebenen Gewichten intern berechnet. Die explizite Gewichtung gibt dem Nutzer die Freiheit, sowohl atomare Terme als auch Teilanfragen explizit mit einem Gewicht zu versehen. Eine Überführung beider Alternativen ineinander ist möglich und notwendig für die Evaluierung einer komplexen Anfrage mittels einer gewichteten Multi-Level Scoring-Funktion.

Die Anfrageoptimierung, welche auf logischen Transformationsregeln basiert, ist in einem Retrieval System wesentlich. Es wurde darauf eingegangen, wie Multi-Level-gewichtete Anfragen logisch umgeformt werden können. Dafür wurden Transformationsregeln angegeben, die eine logische Umformung von gewichteten, komplexen Anfragen erlauben. Die Evaluierung der gewichteten Anfragen erfolgt hier auf Basis der gewichteten Scoring-Funktionen von Fagin und Wimmers. Wir haben gezeigt, wie die Gewichte bei einer distributiven und assoziativen Umformung einer gewichteten, komplexen Anfrage modifiziert werden müssen, so dass sowohl für die ursprüngliche als auch für die transformierte Anfrage stets gleiche Gesamtscores ermittelt werden.

Für unsere weitere Arbeit stehen weitere Untersuchungen zur logischen Umformung von gewichteten, komplexen Anfragen im Vordergrund. Ferner interessiert uns, inwieweit eine Vereinfachung der Formeln zur Berechnung der Gewichte möglich ist. Ein weiteres Ziel ist es, die Transformationsregeln für gewichtete Anfragen in ein Retrieval System einzubetten, um dadurch eine Anfrageoptimierung implementieren zu können.

Literaturverzeichnis

[Boo78] A. Bookstein. On the perils of merging Boolean and weighted retrieval systems. *Journal of the American Society for Information Science, 29*, Seiten 156–158, 1978.

[CBGM97] C. Carson, S. Belongie, H. Greenspan, und J. Malik. Region-based image querying. In *Proc. of the IEEE Workshop CVPR '97 Workshop on Content-Based Access of Image and Video Libraries, Puerto Rico*, Seiten 42–49, 1997.

[DP86] D. Dubois und H. Prade. Weighted Minimum and Maximum Operations in Fuzzy Set Theory. *Information Science 39*, Seiten 205–210, 1986.

[FG01] N. Fuhr und K. Gro'sjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proceedings of the 24th Annual International Conference on Research and development in Information Retrieval, ACM, New York*, Seiten 172–180, 2001.

[FW97] R. Fagin und E.L. Wimmers. Incorporating User Preferences in Multimedia Queries. In *Proc. 6th International Conference on Database Theory*, Seiten 247–261. Springer-Verlag, LNCS 1186, Delphi, 1997.

[FW00] R. Fagin und E.L. Wimmers. A Formula for Incorporating Weights into Scoring Rules. *Theorectical Computer Science*, 239:309–338, 2000.

[HV01] E. Herrera-Viedma. An Information Retrieval System with Ordinal Linguistic Weighted Queries Based on Two Weighting Semantics. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9:77–88, 2001.

[ORC+98] M. Ortega, Y. Rui, K. Chakrabarti, K. Porkaew, S.Mehrotra, und T.S. Huang. Supporting Ranked Boolean Similarity Queries in MARS. *Knowledge and Data Engineering*, 10(6):905–925, 1998.

[Pas99] G. Pasi. A logical formulation of the Boolean model and of weighted Boolean models. In *Proceedings of the Workshop on Logical and Uncertainty Models for Information Systems (LUMIS 99), University College London, Inghilterra*, 1999.

[SFW83] G. Salton, E.A. Fox, und H. Wu. Extended Boolean Information Retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.

[SSnt] N. Schulz und I. Schmitt. Logical Transformation Rules for Complex Weighted Queries. *Preprint, Otto-von-Guericke Univeristät, Magdeburg*, erscheint.

[SSS02] I. Schmitt, N. Schulz, und G. Saake. Multi-Level Weighting in Multimedia Retrieval Systems. In *Proceedings of the 2nd Int. Workshop on Multimedia Data Document Engineering (MDDE'02), Prague, Czech Republic*, Seiten 353–364. Springer-Verlag, LNCS 2490, 2002.

[Sun98] S.Y. Sung. A Linear Transform Scheme for Combining Weights into Scores. *Technical Report, Rice University*, 1998.

[WK79] W. G. Waller und D. H. Kraft. A mathematical model for a weighted Boolean retrieval system. *Information Processing and Management*, 15(5):235–245, 1979.

[Yag87] R. R. Yager. A note on weighted queries in information retrieval systems. *Journal of the American Society for Information Science,38*, Seiten 23–24, 1987.

[Zad65] L. Zadeh. Fuzzy Sets. *Information and Control*, 8:338–353, 1965.

Konstruktion von Featureräumen und Metaverfahren zur Klassifikation von Webdokumenten

Stefan Siersdorfer, Sergej Sizov
{*siersdorfer, sizov*}*@cs.uni-sb.de*

Datenbanken und Informationssysteme
Universität des Saarlandes
66123 Saarbrücken, Deutschland
http://www-dbs.cs.uni-sb.de

Abstract: Dieses Papier befasst sich mit der automatischen Klassifikation von Webdokumenten in eine vorgegebene Taxonomie. Wir betrachten dabei vektorbasierte Verfahren des maschinellen Lernens am Beispiel von SVM (Support Vector Machines). In diesem Papier beschreiben wir Möglichkeiten zur Generierung von Featurevektoren unter Berücksichtigung der Besonderheiten von Webdokumenten für solche Verfahren. Weiterhin untersuchen wir die Berechnung von Metaresultaten aus den partiellen Klassifikationsergebnissen.

1 Einführung und Grundlagen

1.1 Problemstellung

Die Klassifikation von Webinhalten gehört zu den wichtigen Aufgaben des Web Mining. Konventionelle Klassifikationsstrategien basieren auf Verfahren des maschinellen Lernens und verwenden Term-basierte Featurevektoren bei Aufbau und Anwendung des Klassifikationsmodells. Dabei bleiben weitere Aspekte (Umgebung von Webdokumenten, strukturelle Besonderheiten etc.) typischerweise unberücksichtigt.

Diese Arbeit betrachtet unterschiedliche Verfahren zur Generierung von Featurevektoren und deren Zusammenspiel im Rahmen von Meta-Klassifikationsstrategien.

1.2 Dokumentverarbeitung

Um das Klassifikationsverfahren anwenden zu können, müssen wir Dokumente zunächst in Vektoren transformieren. Wir verarbeiten Dokumente in folgenden 3 Schritten mit im Information Retrieval üblichen Methoden:

1. Parsen des Dokuments
2. Elimination von Stoppwörtern
3. Reduktion der Terme auf ihre Stammformen. Wir verwenden den Stemming-Algorithmus nach Porter [Pora, Porb].
4. Berechnung der Feature-Vektoren. Entsprechende Verfahren werden in Kapitel 2 näher betrachtet.

1.3 Hierarchische Klassifikation

Wir betrachten den Taxonomiebaum der benutzerspezifischen Themen (Abbildung 1). Jedem Knoten ist eine Menge von intellektuell bestimmten Trainingsdokumenten zugeord-

net. Für alle Knoten außer ROOT berechnen wir nun einen SVM-Klassifikator. Für eine Klasse K betrachten wir dabei die Dokumente aus K als Positivbeispiele, die Dokumente aus den Nachbarklassen von K mit dem selben Vater wie K (Gegnerklassen von K) als Negativbeispiele. Ein neues Dokument können wir nun klassifizieren, indem wir den Baum ausgehend von der Wurzel traversieren und die Klassifikationen mittels der einzelnen Knotenmodelle durchführen. Wird ein Dokument dabei ausgehend von einer Oberkategorie in mehrere Unterkategorien klassifiziert, so wählen wir den Knoten mit der höchsten Klassifikationskonfidenz (im Falle von SVM: der größte Abstand von der Hyperebene). Wird das Dokument in keine der Unterkategorien positiv klassifiziert, so ordnen wir das Dokument einer Sonderklasse OTHERS zu. Wir verwenden linea-

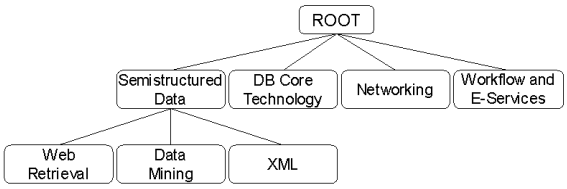


Abbildung 1: Beispieltaxonomie

re Support Vector Machines (SVM) [Bur98, Vap98] als themenspezifischen Klassifikator. Diese Methode hat sich als effizient und effektiv für die Textklassifikation erwiesen (siehe [DC00, CD00, Joa98]). Das Training besteht dabei in der Berechung einer trennenden Hyperebene im m-dimensionalen Featureraum, die eine Menge von positiven Trainingsbeispielen von einer Menge von negativen Beispielen trennt (Abbildung 2). Die Hyperebene kann in der Form $\vec{w}\vec{x} + b = 0$ beschrieben werden. Die Parameter $\vec{w}$ und b der optimalen Hyperebene werden bei SVM nun so bestimmt, dass der Euklidische Abstand δ der nächstgelegenen Vektoren von der Hyperebene maximiert wird:

$$C_i \frac{1}{||\vec{w}||} (\vec{w}\vec{x}_i + b) \geq \delta \tag{1}$$

für alle i, wobei $\vec{x}_i \in \mathbf{R}^m$ das i-te Trainingsbeispiel ist und $C_i \in \{1, -1\}$ beschreibt, ob x_i ein positives ($C_i = 1$) oder ein negatives ($C_i = -1$) Beispiel ist.

1.4 Featureselektion

Featureselektion reflektiert die Annahme, dass einige Terme irrelevant für die Klassifikation sind und daher bei der Berechnung von Featurevektoren ignoriert werden können. Der Featureselektionsalgorithmus sollte für eine gegebene Klasse die charakteristischsten Features auswählen. Ein gutes Feature sollte eine Klasse gut von seinen Gegnerklassen unterscheiden. Daher sollte Featureselektion themenspezifisch sein: sie wird individuell für jede Klasse des Ontologiebaums durchgeführt.

Wir verwenden das Mutual Information (MI)- Kriterium für themenspezifische Features. Diese Technik, die eine Spezialfall von Kreuzentropie oder Kullback-Leibler Divergenz [MS99] ist, ist als eine der effektivsten Methoden bekannt [YP97]. Die MI-Gewichtung eines Terms X_i und einer Klasse V_j ist definiert durch:

$$MI(X_i, V_j) = P[X_i \wedge V_j] \log \frac{P[X_i \wedge V_j]}{P[X_i]P[V_j]} \tag{2}$$

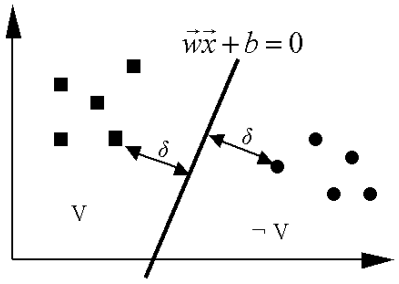


Abbildung 2: Separierende Hyperebene eines linearen SVM-Klassifikators

Mutual Information kann als Maß dafür interpretiert werden, wie stark sich die gemeinsame Verteilung der Features X_i und der Klassen V_j von einer hypothetischen Verteilung unterscheiden, in denen Features und Klassen unabhängig voneinander sind.

2 Konstruktion von Featureräumen

2.1 Einzeltermfeatureräume

Zu den einfachsten Möglichkeiten zur Konstruktion von Featurevektoren gehören Verfahren, die auf den relativen Häufigkeiten von Einzeltermen in einem Dokument basieren. Hier beschreiben wir zwei einfache Varianten zur Konstruktion von Einzeltermvektoren mit Hilfe von MI.

Strategie der klassenweise besten Terme Zur Berechung des Vektors zu einem Dokument D bezüglich Klasse K wählen wir durch MI-Selektion eine Menge von m charakteristischen Termen $\{t_1, \ldots, t_m\}$. Wir erzeugen den Featurevektor aus den relativen Häufigkeiten $tf(D, t_i)$ der t_i in D:

$$\big(tf(D, t_1), \ldots, tf(D, t_m)\big) \tag{3}$$

Da die Terme aus $T(K)$ K nach Konstruktion besonders gut von ihren Gegnerklassen beim Training unterscheiden, werden die Vektoren zu Dokumenten aus K eher *dicht* besetzt sein, die Vektoren aus den Nachbarklassen hingegen eher *dünn* besetzt sein. Dicht besetzt soll hier heißen, dass viele Komponenten einen relativen großen Wert > 0 haben. Dünn besetzt heißt, dass viele Komponenten den Wert 0 haben (keine Terme im Dokument vorhanden) bzw. eher kleinere Werte für die relative Häufigkeit besitzen. Dadurch erreichen wir eine gute Trennbarkeit.

Strategie der Kontrastterme Die Idee der *Konstraststrategie* für Einzelterme ist, charakteristische Terme für Nachbarklassen bei der Modellbildung für eine Klasse K mit einzubeziehen. Diese Terme der Nachbarklassen bezeichnen wir als *Kontrastterme*. Die Featurevektoren konstruieren wie analog zu den Vektoren aus den klassenweise besten Einzeltermen. Ziel ist es, Vektoren zu konstruieren, die mehrere unterschiedlich dicht besetzte Bereiche haben. Dabei gibt es Bereiche von Komponenten, die bei Positivvektoren typischerweise dicht besetzt, bei Negativvektoren hingegen dünn besetzt sind und umgekehrt. Dadurch wollen wir wieder eine bessere Trennbarkeit erreichen.

2.2 Paarstrategien

Grundideen In vielen Dokumenten tauchen zusammengehörige Begriffe auf (z.B. network und protocol). Außerdem existieren viele Begriffe, die in bestimmten Kontextbereichen häufig gemeinsam auftreten. Wir können also versuchen, Textterme zu Tupeln (*n-Grammen*) zusammenzufassen.

Der Ansatz, alle n-Tupel im Termraum zu betrachten scheitert, da deren Anzahl exponentiell ist. Wir wollen uns im Folgenden auf $n = 2$, also auf *Paare* beschränken. Dies rechtfertigen wir zum einen durch die Beobachtung, dass wir signifikante Häufigkeiten in Dokumenten seltener bei höher dimensionierten Tupeln finden können. Außerdem impliziert eine Korrelation von n Termen auch eine Korrelation der $\binom{n}{2}$ daraus bildbaren Paare.

Schränken wir die Paarmenge nicht ein, ist deren Anzahl quadratisch in der Größe des Termraums. Die Bildung von Vektoren und die Anwendung des SVM-Verfahrens wären somit mit einem erheblichen Berechnungsaufwand verbunden. Ein weiteres Problem ist die Existenz von verschiedenen *Kontextbereichen* in Dokumenten. Ohne Einschränkungen erhalten wir somit Paare mit Termen aus jeweils unterschiedlichen Sinnabschnitten.

Paarräume Formal stellen wir Dokumente D als Menge von Tupeln (t, pos) dar, wobei t der Term selbst und pos die *Position* des Terms im Dokument ist. Die für uns relevanten Informationen über ein Termpaar können wir in der Form

$$(t_a, t_b, pos_a, pos_b) \tag{4}$$

darstellen. t_a und t_b sind dabei die Terme, pos_a und pos_b sind die Positionen von t_a bzw. t_b im betrachteten Dokument.

Es soll $t_a \neq t_b$ gelten, d.h. wir assoziieren keine Terme mit sich selbst. Wir behandeln Paare symmetrisch , d.h. wir machen keinen Unterschied zwischen (t_a, t_b) und (t_b, t_a). Um eine kanonische Form zu erhalten, legen wir fest, dass gelten soll: $t_a <_{lex} t_b$.

Für eine Klasse K können wir den Paarraum einschränken, indem wir nur Paare über einer Menge $\tilde{T}(K) = \{t_1, \ldots, t_q\}$ von Termen bilden, die wir durch eine der beschriebenen Einzeltermstrategien bestimmen.

Bei der *Sliding-Window* -Strategie betrachten wir nur Paare aus Termen, die sich innerhalb eines gedachten Fensters der Größe dis befinden, dass wir über den Text eines Dokumentes bewegen:

$$\forall (t_a, t_b, pos_a, pos_b) \in D : |pos_a - pos_b| < dis \tag{5}$$

Insgesamt definieren wir die Menge $P(K)$ der für die Modellierung von K relevanten Paare bei Verwendung der Trainingsdokumentmenge $Tr(K)$:

$$\begin{aligned} P(K) \quad := \quad \Big\{ (t_a, t_b) \quad \Big| \quad & t_a \neq t_b \quad \wedge \quad t_a <_{lex} t_b \\ \wedge \quad & \Big(\exists D \in Tr(K), pos_a, pos_b : \\ & (t_a, t_b, pos_a, pos_b) \in D \wedge |pos_a - pos_b| < dis \Big) \Big\} \end{aligned} \tag{6}$$

bzw. bei Einschränkung der Einzelterme auf $\tilde{T}(K)$ die Menge $\tilde{P}(K)$:

$$\tilde{P}(K) = \{(t_a, t_b) \in P(K) | t_a \in \tilde{T}(K) \wedge t_b \in \tilde{T}(K)\} \tag{7}$$

Wir können für ein Paar (t_a, t_b) nun einfach die Anzahl der Tupel (t_a, t_b, pos_a, pos_b) in einem Dokument D zählen. Dann kann allerdings z.B. folgende Situation auftreten: Gehen wir von einem Term-Positionspaar (t_a, pos_a) in D aus, so kann ein Term-Positionspaar (t_b, pos_b) und (t_b, pos'_b) existieren, so dass $pos'_b > pos_b$. Das Paar (t_a, t_b) würde also doppelt gezählt. Die Vermutung liegt jedoch nahe, dass t_a eher mit dem näher gelegenen der beiden t_b assoziiert werden kann. Daher zählen wir in solchen Fällen im Dokument weiter auseinander liegende Paare nicht mit. Diese bezeichnen wir als *Pseudopaare* in D, $Pseudo(D)$ als deren Menge.

Wir erweitern den Häufigkeitsbegriff auf Paare. Für $(t_a, t_b) = p \in P(K)$ definieren wir die absolute Häufigkeit in Trainingsdokument D als:

$$h_{abs}(p, D) = \Big|\{(w_s, w_t, k, l) \in D \mid w_s = t_a \wedge w_t = t_b \wedge (w_s, w_t, k, l) \notin Pseudo(D) \wedge |k - l| < dis\}\Big| \quad (8)$$

Die Komponenten des Featurevektors für D bestehen nun aus den daraus abgeleiteten relativen Häufigkeiten der Paare aus $\tilde{P}(K)$.

Wir können die Menge der Paare noch weiter durch Featureselektion für *Paare* eingrenzen. Analog zu Einzeltermen verwenden wir dazu die Strategie der klassenweise besten *Paare* und die Strategie der Kontrast*paare*.

2.3 Anchorterme

Anchortexte sind die Texte zu Links in HTML-Seiten. Häufig liefern sie kurze und prägnante Beschreibungen der Seiten, auf die die entsprechenden Links zeigen. Wir wollen daher auch *Anchorterme* zur Modellbildung und Klassifikation von Seiten verwenden. Dabei wollen wir nur die Anchortexte von Seiten betrachten, die auf die interessierende Seite D zeigen. Diese Links beziehen sich unabhängig vom Inhalt der Seite *von* der verwiesen wird auf die Seite D.

Die resultierende (Multi)-Menge der Anchorterme können wir nun im Prinzip wie ein virtuelles Einzeltermdokument behandeln. Dabei ist eine Elimination von Stoppwörtern notwendig um Artefakte wie *click here* zu vermeiden. Das Hauptproblem ist allerdings dass die Mengen der Anchorterme häufig sehr klein sind. MI-Featureselektion, die statistisch begründet ist, greift daher oft nur schlecht. Daher wird eine reine Anchortermstrategie wahrscheinlich wenig aussichtsreich sein. Trotzdem erlaubt dieser Ansatz sinnvolle Kombinationen mit anderen Strategien (siehe Abschnitt 2.5). Dies unterscheidet unsere Vorgehensweise von verwandten Klassifikationsmethoden für Webdaten [CJT01].

2.4 Dokumentvereinigung

Die Idee bei der *Dokumentvereinigung* besteht darin, die Vorgänger und Nachfolger eines Dokuments D zur Konstruktion von Featurevektoren mit einzubeziehen. Nachbardokumente haben häufig mit dem Inhalt von D und damit der Kategorie von D zu tun. Die Qualität von Nachbardokumenten lässt daher oft Rückschlüsse auf die Qualität von D zu.

Der einfachste Weg wäre nun die Gesamtheit der zu vereinigenden Dokumente wie ein einziges zu behandeln und die entsprechenden Einzeltermstrategien aus Abschnitt 2.1 anzuwenden. Leider führt diese Strategie in primitiver Form zu unbefriedigenden Klassifikationsergebnissen [CJT01]. Wir wollen allerdings vermeiden, dass Dokumente, die groß sind relativ zu anderen Dokumenten der Vereinigung, eine unverhältnismäßige Rolle spielen. Dies können wir durch einfache Termgewichtungen nicht erreichen. Statt die Terme

der Dokumente zu gewichten, können wir dies aber auch mit den Dokumenten selbst tun. Wir bilden also einen gewichteten Mittelwert über die tf-Werte der Einzeldokumente.

Zum Bilden eines SVM-Modells ist die Dokumentvereinigung nur bedingt geeignet, da nur in sehr seltenen Fällen die Klasse eines Dokuments D mit den Klassen *aller* Nachbarn übereinstimmen wird. Somit ist meist auch unter Einbeziehung von Featureselektion eine Verschlechterung der Modellqualität zu erwarten. Modellierung mittels Dokumentvereinigung sollte höchstens als Ergänzung im Rahmen von Metaverfahren (siehe Kapitel 3) angewendet werden.

2.5 Kombinationsräume

Einzelterme und Paare Bei der Bildung reiner Paarvektoren können Informationen über separat auftretende Einzelterme schnell verloren gehen. Bei den Einzeltermstrategien bleibt dagegen die Anordnung der Terme im Dokument unberücksichtigt. Aus diesen Gründen ist es nahe liegend, Einzeltermvektoren und Paarvektoren zu kombinieren.

Sei also eine zu untersuchende Klasse K, eine Menge $\tilde{T} = \{t_1, \ldots, t_n\}$ von relevanten Einzeltermen und eine Menge $P(K) = \{p_1, \ldots, p_m\}$ von relevanten Paaren gegeben. Dann können wir für ein Dokument D Featurevektoren der folgenden Form bilden:

$$(tf(t_1, D), \ldots, tf(t_n, D), \ldots, tf(p_1, D), \ldots, tf(p_m, D)) \tag{9}$$

Anchors und Terme Wir haben schon erwähnt, dass Anchortermvektoren für sich gesehen wohl wenig Sinn machen. Als Gründe hatten wir dünne Besetzung und unterschiedliche Verfügbarkeit genannt. Einzeltermvektoren und Anchortermvektoren analog zum oberen Abschnitt zusammenzulegen, scheint aus demselben Grund wenig viel versprechend. Wir beziehen die Anchorterme stattdessen linear mit einer Gewichtung in die Berechnung der relativen Häufigkeiten der Dokumentterme (Einzelterme) mit ein.

3 Metaverfahren

Die Idee von *Metaverfahren* ist es nun, mehrere Verfahren auf eine gegebene Problemstellung anzuwenden. Diese Idee ist verwandt mit der Technik der zusammengesetzten Kernel [JCST01]. Es wurde allerdings gezeigt, dass diese Technik bei dünn besetzten Featureräumen mit vielen Stützvektoren (speziell für Webdaten) und stark unterschiedlicher Präzision einzelner Klassifikationsstrategien keine Verbesserung der resultierenden Klassifikationsgüte erzielt. Statt dessen versuchen wir, auf die *Ergebnisse* einzelner Verfahren unser Metaverfahren anzuwenden und ermitteln so das *Metaresultat*.

Sei eine Menge von n unterschiedlichen Verfahren $V = \{v_1, \ldots, v_n\}$ zur Konstruktion von Termfeatureräumen und damit zur Klassifikation von Dokumenten gegeben. $Res(v_i, D, K)$ bezeichne das Resultat der Klassifikation von Dokument D bei Betrachtung der Klasse K und Verwendung des Verfahrens v_i. Dabei soll $Res(v_i, D, K) = +1$ gelten, falls D durch v_i in K klassifiziert wurde, $Res(v_i, D, K) = -1$ sonst.

Wir können den Verfahren aus V nun eine Gewichtung $w(v_i) \in \mathbf{R}_+$ zuordnen, die ein Maß für die Bedeutung sein soll, die wir dem Verfahren zuordnen. Die Qualität eines Verfahrens können wir z.B. durch die Auswertung von Experimenten abschätzen.

Damit können wir eine *Metaresultatsfunktion* $Meta(V, D, K)$ zur Klassifikation eines Dokumentes D bei Betrachtung einer Klasse K, Anwendung der Verfahrensmenge V und

Schranken $t_1, t_2 \in \mathbf{R}$ mit $thres_1 \geq thres_2$ de nieren:

$$Meta(V, D, K) = \begin{cases} +1 & \text{falls } \sum_{i=1}^{n} Res(v_i, D, K) \cdot w(v_i) > t_1 \\ -1 & \text{falls } \sum_{i=1}^{n} Res(v_i, D, K) \cdot w(v_i) < t_2 \\ 0 & \text{sonst} \end{cases} \quad (10)$$

Dabei soll $Meta(V, D, K) = +1$ bedeuten, dass D nach dem Metaverfahren in K liegt, $Meta(V, D, K) = -1$ soll bedeuten, dass D nicht in K liegt. Nimmt die Metaresultatsfunktion den Wert 0 an, so treffen wir keine Aussage über das Klassi kationsergebnis.

Wir wollen drei wichtige Spezialfälle betrachten.

- Einstimmigkeit : Wir verlangen, dass die Resultate aller Verfahren aus V für die Klassi kation von Dokument D bezüglich Klasse K übereinstimmen sollen. Das Metaresultat soll gleich dem Resultat der Verfahren sein. Liefern die Verfahren keine einheitlichen Resultate, so treffen wir keine Aussage über das Klassi kationsergebnis. (Parameterisierung: $w(v_i) = 1 \quad \forall v_i \in V, \quad t_1 = n - 0.5 = -t_2$).
- Mehrheitsentscheid : Wir übernehmen das Resultat, das die Mehrheit der Verfahren aus V liefert, als Metaresultat. ($w(v_i) = 1 \quad \forall v_i \in V, \quad t_1 = t_2 = 0$.)
- Gewichtung durch $\xi\alpha$-Estimatoren: Diese von Joachims für SVM eingeführte Estimatoren $PR^{\xi\alpha}$ für die Präzisionsqualität stellen eine ef ziente Abschätzung für leave-one-out Verfahren zur Verfügung [Joa00]. ($w(v_i) = PR^{\xi\alpha}(v_i) \ \forall v_i \in V, \ t_1 = t_2 = 0$)

Statt bei der Metaresultatsfunktion über ± 1 zu summieren könnte man auch über die jeweiligen Kon denzen (Abstand zu der Hyperebene) summieren. Die Abstände für ein Verfahren, könnte man z.B. normieren, indem man diese durch den mittleren Abstand von der Hyperebne bei Anwendung des Verfahrens auf die Trainingsdaten dividiert.

Je nach Wahl von V und der Parameter erhalten wir unterschiedlich restriktive Verfahren. Diese Restriktivität führt zwar dazu, dass ein bestimmter Anteil an zu klassi zierenden Dokumenten verworfen wird; allerdings zeigen Experimente (siehe Kapitel 4), dass sich die Qualität der Klassi kation der restlichen Dokumente erhöht.

4 Experimente

4.1 Versuchsaufbau

In unseren Experimenten verwendeten wir für die Aquisition der Webdaten den fokussierten Crawler BINGO! [SSTW02] in Verbindung mit der Ontologie 1.3. Als Bookmarks wurden Homepages bekannter Wissenschaftler aus entsprechenden Forschungsbereichen manuell ausgewählt. Die Blattknoten der Ontologie wurden initialisiert mit 9-15 Bookmarks pro Blatt; die gesamte Trainingsbasis enthielt 81 Dokumente.

Der anschließende fokussierte Crawl fand, ausgehend von dieser Startmenge, 2738 positiv klassi zierte Seiten auf Link-Entfernungen zwischen 1 und 7. Der Crawl wurde nach 6 Stunden angehalten; insgesamt wurden dabei ca. 24750 URLs auf 7149 unterschiedlichen Hosts besucht.

Für die Bewertung der Crawling-Iterationen wurde - durch manuelle Auswertung der Ergebnismenge - die Präzision des Klassi kators (Anteil der richtigen positiven Klassi kationsentscheidungen pro Knoten) berechnet. Weitere interessante Kenngrößen sind die Ausbeute (Anzahl der Crawl-Resultate pro Knoten), die Zahl der Archetypen und deren Qualität (Anteil der korrekt ermittelten themenspezi schen Dokumente unter den Archetypen).

Die Auswertung erfolgte für folgende Klassi kationsstrategien:

1. Einfacher Term-basierter SVM Klassi kator ohne Selektion der Features [AllTerms]
2. Term-basierter SVM Klassi kator mit Selektion der Features durch MI (300 klassenweise beste Features [SelTerms(300)];
3. Term-basierter SVM Klassi kator mit Selektion der Features durch MI mit Kontrasttermen (300 beste klassenspezi sche Features plus je 100 beste Features aus jeder Gegnerklasse)[ContrTerms(300,100)];
4. Paar-basierter SVM Klassi kator (Sliding-Window der Gröÿe $SW = 16$) mit Pre-Selektion der Terme durch MI (300 beste Terme für Paarbildung) [Pairs(pre:300,SW:16)];
5. Paar-basierter SVM Klassi kator ($SW = 16$) mit Pre-Selektion der Terme durch MI und Verwendung der Kontrastfeatures (300 Klassenspezi sche Termen plus je 100 pro Gegnerklasse) [Pairs(contr:(300,100),SW:16)];
6. Kombinierter SVM Klassi kator auf Termen und Paaren ($SW = 16$) mit Selektion der Features durch MI (Vorauswahl 300 beste Terme für Paarbildung) [Pairs&Terms(pre:300,SW:16)];
7. Kombinierter SVM Klassi kator auf Termen und Paaren ($SW = 16$) mit Selektion der Kontrastfeatures und Vorauswahl durch MI (300 beste klassenspezi sche Terme plus je 100 Kontrastterme pro Gegnerklasse für Paarbildung) [Pairs&Terms(contr:(300,100),SW:16)];

Auÿerdem bewerteten wir für diese Gruppe der Klassi katoren die Ergebnisse der Meta-Strategien Mehrheitsentscheid und Gewichteter Mehrheitsentscheid nach dem $\xi\alpha$-Gewichtungsschema. Aufgrund des hohen Berechnungsaufwandes, verzichteten wir in unseren ersten Experimenten auf die Auswertung der Strategien auf der Basis der Dokumentvereinigung.

In einer weiteren Experimentserie haben wir die restriktive Meta-Strategie Einstimmigkeit betrachtet.

4.2 Ergebnisse

Die Tabelle 1 zeigt eine Zusammenfassung der Auswertungen für verschiedene Klassi kationstechniken und die Zusammenfassung der Archetyp-Selektion in der ersten Crawling-Iteration für die - mit 10 Bookmarks initialisierte - Klasse *"Root/Semistructured Data/Data Mining"*. Angegeben sind auÿerdem die Schätzung der Präzision durch den $\xi\alpha$-Estimator und der tatsächliche Wert nach der manuellen Auswertung.

Verfahren	Klassifiziert	Korrekt	$\xi\alpha$	Präzision	Archetypen	Korrekt	Präzision
AllTerms	229	181	0.71	0.79	32	27	0.84
SelTerms(300)	197	162	0.78	0.82	29	25	0.86
ContrTerms(300,100)	184	158	0.77	0.85	31	24	0.77
Pairs(pre:300,SW:16)	208	152	0.74	0.73	27	23	0.85
Pairs (contr:(300,100),SW:16)	169	149	0.81	0.88	33	28	0.84
Pairs&Terms (pre:300,SW:16)	155	141	0.84	0.91	42	32	0.76
Pairs&Terms (contr(300,100), SW16)	142	122	0.85	0.86	35	31	0.88
Majority	98	92	-	0.94	15	14	0.93
Mutual	95	91	-	0.96	19	17	0.89

Tabelle 1: Präzision der Klassifikationsverfahren und der Archetyp-Selektion

Die Ergebnisse zeigen die Vorteile der Meta-Klassi kationsstrategien. Die parallele Betrachtung verschiedener Dokumenteigenschaften ermöglicht gute Präzision und Qualität

der erkannten Archetypen auch bei mäßiger Präzisionserwartung einzelner Klassifikationsmethoden.

Natürlich darf man nicht vergessen, dass höhere Präzision der Metaverfahren mit einem deutlich höheren rechnerischen Aufwand verbunden ist und somit primär in besonders präzisions-sensitiven Anwendungen benutzt werden sollte, z.B. beim Re-Training des fokussierten Crawlers oder für die Revision der besten Suchergebnisse. In diesen Fällen kann die Menge der zu validierenden Dokumente entsprechend begrenzt werden.

Tabelle 2 zeigt die Ergebnisse des Meta-Verfahrens Einstimmigkeit. Wir haben hier in drei Experimenten unterschiedliche Kombinationen von Einzelverfahren über einer allgemeineren und fachen Hierarchie getestet, die aus den Kategorien *Business*, *Computers and Internet*, *Sports* und *Health* bestand. Wir konnten, bei relativ moderater Reduktion der Dokumentmenge, eine deutliche Erhöhung der Präzision gegenüber dem besten Einzelverfahren erzielen.

Experiment	Klassifiziert	einh. Klassifiziert	Prec bestes Einzelverf.	Prec Einstimmigkeit
1	242	183	0.95	0.99
2	1012	668	0.87	0.96
3	757	694	0.91	0.97

Tabelle 2: Präzisionsverbesserung bei dem restriktiven Meta-Verfahrens "Einstimmigkeit" mit Dokumentreduktion gegenüber dem jeweils besten Einzelverfahren

Eine interessante Beobachtung hinsichtlich der vorgestellten Metastrategien bestand in der effektiven Erkennung von Dokumenten, die keiner der Klassen der Taxonomie zugeordnet werden konnten. Nur 166 von 614 nicht eindeutigen oder falschen Dokumente (ca. 27 %) wurden z.B. in Experiment 2 durch das Metaverfahren einer der Klassen zugeordnet. Somit sind Metaverfahren - insbesondere in Kombination mit anderen Techniken - für die Grob lterung der irrelevanten Dokumente gut geeignet.

5 Fazit und Ausblick

Fazit In diesem Papier haben wir unterschiedliche Methoden zur Konstruktion von Featureräumen für vektorbasierte maschinelle Lernverfahren (am Beispiel von SVM) zur Klassifikation von Webdokumenten kennen gelernt. Jeder dieser Ansätze untersucht und kombiniert unterschiedliche Aspekte der betrachteten Dokumente wie z.B. Anordnung der Terme im Dokument oder Dokumentumgebung. Unsere Experimente zeigen, dass die meisten dieser Verfahren schon für sich gesehen Sinn machen.

Durch Anwendung von Metaverfahren konnten wir noch eine deutliche Erhöhung der Klassifikationspräzision erzielen. Eine Verbesserung der Präzision haben wir insbesondere durch eine restriktive Variante von Metaverfahren erreicht, bei der wir über einen Teil der Dokumente keine Aussage treffen, bei den übrigen Dokumenten aber bessere Ergebnisse erzielen. Interessant ist dies u.a. für ein unüberwachtes Lernsystem, bei dem die ursprüngliche Trainingsdatenmenge automatisch um neu klassifizierte Dokumente ergänzt wird. Bei der Fülle von Inhalten im WWW können wir nicht davon ausgehen, dass unsere Trainingsdaten alle Themenbereiche erfassen (Häufig sind wir auch nur an speziellen Taxonomien interessiert). Diese Themenbereiche lassen sich auch durch Vorschalten von Hilfshierarchien und Hinzunahme von Sonderklassen für sonstige Inhalte (was im Übrigen wieder mit teurem intellektuellen Aufwand verbunden ist) nicht völlig abdecken. Ein automatischer Crawler wird jedoch mit hoher Wahrscheinlichkeit mit solchen Dokumenten umgehen müssen. Dies unterscheidet die hier gegebene Aufgabenstellung von klassischen

Experimenten zu Textklassifikation über Dokumenten, die alle zu wohldefinierten Trainingskategorien gehören (z.B. bei den typischen Reuter-Benchmarks). Experimente zeigen, dass Metaverfahren einen ersten Ansatz bieten, mit dieser Problematik umzugehen.

Ausblick Wir betrachten das vorgestellte Framework der Klassifikationsstrategien als eine Basis für weiterführende Untersuchungen und Experimentreihen. Zu den wichtigen, bisher unzureichend untersuchten, Aspekten dieser Arbeit gehört die Fehleranfälligkeit des Verfahrens (Auswirkung von falschen Trainingsbeispielen, z.B. inkorrekten Archetypen, auf das Gesamtergebnis des fokussierten Crawls). Auch die Frage des optimalen Re-Trainings (Zeitpunkt der Auslösung, Anzahl der verwendeten Archetypen und Features) soll näher untersucht werden. Eine besondere Aufmerksamkeit ist dabei der Erzeugung von Feature-Mengen (Typ, Anzahl der Features verschiedener Arten) zu schenken, um den unnötigen Aufwand in leicht separierbaren Fällen zu vermeiden. Zum Beispiel, die aufwendigeren Strategien (Termpaare, Dokumentvereinigung etc.) können einbezogen werden, falls die geschätzte Präzision des Term-basierten Klassifikators nicht ausreichend ist. Zu unseren weiteren Zielen gehört die Entwicklung eines formalen Modells der vorgestellten Metaverfahren, um deren Funktionsweise besser zu verstehen sowie allgemeine Leistungsgrenzen genauer abschätzen zu können. Ferner soll das vorgestellte Framework in Zukunft durch Clustering-Algorithmen für nicht eindeutig klassifizierte Dokumente erweitert werden, um unvollständige benutzerdefinierte Ontologien durch automatisch generierte Gruppen von thematisch verwandten Dokumenten dynamisch erweitern zu können.

Literatur

[Bur98] C.J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2), 1998.

[CD00] H. Chen and S. Dumais. Bringing Order to the Web: Automatically Categorizing Search Results. *ACM CHI Conference on Human Factors in Computing Systems*, 2000.

[CJT01] S. Chakrabarti, M.M. Joshi, and V.B. Tawde. Enhanced Topic Distillation using Text, Markup Tags, and Hyperlinks. *ACM SIGIR Conference*, 2001.

[DC00] S. Dumais and H. Chen. Hierarchical Classification of Web Content. *ACM SIGIR Conference*, 2000.

[JCST01] T. Joachims, N. Cristianini, and J. Shawe-Taylor. Composite Kernels for Hypertext Categorisation. *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

[Joa98] T. Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. *European Conference on Machine Learning (ECML)*, 1998.

[Joa00] T. Joachims. Estimating the generalization performance of an SVM efficiently. *European Conference on Machine Learning (ECML)*, 2000.

[MS99] C.D. Manning and H. Schuetze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[Pora] M. Porter. An algorithm for suffix stripping. *Automated Library and Information Systems*, 14(3).

[Porb] M. Porter. The Porter Stemming Algorithm. *http://www.tartarus.org/ martin/PorterStemmer/index.html*.

[SSTW02] S. Sizov, S. Siersdorfer, M. Theobald, and G. Weikum. The BINGO! focused crawler: From Bookmarks to Archetypes. *IEEE Computer Society International Conference on Data Engineering (ICDE), San Jose, California*, 2002.

[Vap98] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[YP97] Y. Yang and O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. *International Conference on Machine Learning (ICML)*, 1997.

Ähnlichkeitssuche in Musik-Datenbanken mit Hilfe von Visualisierungen

Dirk Habich
Informatik, Uni Halle, Deutschland
habich@informatik.uni-halle.de

Alexander Hinneburg
Informatik, Uni Halle, Deutschland
hinneburg@informatik.uni-halle.de

Abstract: Ähnlichkeitssuche in Datenbanken wurde bisher in vielen Bereichen erfolgreich angewendet. Jedoch gibt es vergleichsweise wenige Arbeiten, die sich mit Ähnlichkeitssuche in Musikdaten beschäftigen. Da derzeit immer mehr Musik über das Internet verfügbar wird, stößt dieser Bereich zunehmend bei vielen Anwendungsgruppen auf reges Interesse. Bisherige Suchverfahren lassen sich aber nur im beschränkten Maße an die vielfältigen Anwendungsszenarien anpassen. Meist läßt sich dies nur über die Auswahl einer abstrakten Metrik realisieren. Jedoch ist es ein ungelöstes Problem, wie der Benutzer dem Suchsystem mitteilen kann, welche Aspekte bei der Suche für eine Aufgabe relevant sind.

Diese Arbeit stellt einen Ansatz vor, der versucht, die sogenannte *semantische Lücke* zwischen Benutzer und System durch einer Kombination aus konventioneller Ähnlichkeitssuche mit interaktiven Visualisierungen zu überbrücken. Dafür wurde eine neue Feature-Extraktionsmethode für Musikdaten entwickelt, die gleichzeitig für eine Visualisierung geeignet ist. Die abgeleitete Visualisierung beschreibt statistische Eigenschaften des Musikstücks. Mit Hilfe der Visualisierung kann die Ähnlichkeit zweier Musikstücke auf herkömmliche Weise akustisch, aber auch visuell bewertet werden. Der visuelle Weg ist viel schneller als das akustische Durchhören verschiedener Resultate und ermöglicht so die Verwendung von *Relevance Feedback*, mittels dessen das System sich iterativ an die Vorstellungen des Benutzers anpassen kann. Wir haben unseren Ansatz mit einer bekannten Methode für Musik-Ähnlichkeitssuche verglichen und demonstrieren die Effektivität anhand von Anwendungsbeispielen.

1 Einführung

Ähnlichkeitssuche in großen Datenbanken ist ein wichtiges Forschungsgebiet mit vielfältigen Anwendungsgebieten. Hier ist unter anderem Ähnlichkeitssuche in Bild-Datenbanken, in Datenbanken mit geometrischen, geographischen und CAD-Objekten und in Dokument-Datenbanken zu nennen. Jedoch relative wenige Arbeiten wurden bisher über Musikähnlichkeitssuche veröffentlicht.

1.1 Musik Retrieval

Die meisten Arbeiten über Audiodaten kommen aus dem Bereich Spracherkennung. Es gibt einige Versuche, die dort entwickelten Verfahren auf Musikähnlichkeitssuche zu übertragen. Eine einfache Übertragung ist jedoch nur beschränkt möglich, da sich Sprache und Musik in ihren Eigenschaften (Frequenzen, Rhythmus, usw.) stark unterscheiden [Sch00].

Publizierte Arbeiten über Musikdaten umfassen unter anderem die Bereiche Analyse, Instrumenterkennung, Klassifikation und inhaltsbasierte Suche. Musikanalyse beschäftigt sich mit Rhythmus-, Melodie- und Harmonieerkennung [Rap01, DC01, BB01] ebenso wie mit der Frage, welche Eigenschaften für Musik-Ähnlichkeit wichtig sind [YK99]. Instrumenterkennung hat das Ziel in einem Musikstück die verwendeten Instrumente zu isolieren [HBABS00].

Musik-Klassifikation untersucht Methoden, die das automatische Einordnen von elektronisch gespeicherten Musikstücken nach verschiedenen Genres erlauben. Hier wurden verschiedene Trainingsansätze vorgeschlagen, bei denen die verschiedenen Klassen von den Verfahren selbst bestimmt werden [WBKW96]. Pampalk, Rauber und Merkl [PRM02] beschreiben eine Methode, die nach dem Lernschritt die Beziehungen zwischen den Klassen mittels einer abstrakten Karte visualisiert.

Ein wichtiger Aspekt der inhaltsbasierten Ähnlichkeitssuche auf Musikdaten ist das Problem der Anfragegenerierung. Die beiden vorgeschlagenen Ansätze sind erstens: die Anfrage wird vom Benutzer ins Mikrofon gesummt [KNS+00]; zweitens: es wird ein Anfragemusikstück in elektronischer Form dem Suchsystem präsentiert. Der erste Ansatz eignete sich besser für Musikstücke mit einfachem Melodieverlauf, wie zum Beispiel Volkslieder und ist oft auf Musikstücke im MIDI-Format (Musical Instrument Digital Interface) beschränkt, bei dem die verschiedenen Tonsequenzen separat in symbolischer Form abgespeichert werden. Der Großteil der elektronisch verfügbaren Musik liegt jedoch in digitalen Audio-Formaten wie WAV oder MP3 vor, die mittels Sampling aus analogen Aufnahmen gewonnen werden und nur die überlagerten Signale repräsentieren. Eine befriedigende Umwandlung von stark überlagerten Musiksignalen in einem Audio-Format in das MIDI-Format ist trotz sehr hohem Aufwand kaum möglich.

Um Musikstücke im Audioformat miteinander zu vergleichen, werden die Audiodaten in mehrdimensionale Vektoren transformiert, in denen Eigenschaften des Stücks zusammengefaßt sind. Für die Umwandlung können die Audio-Daten von zwei Seiten betrachtet werden. Die erste Sichtweise nutzt meßbare physikalischen Größen, wie Amplitude oder Frequenz. Die als Fourier-Transformation [Bri74] bekannte Zerlegung des Amplituden-Signals in Sinus-Wellen verschiedener Amplitude und Wellenlänge ist ein wichtiger Analyseschritt, der in ähnlicher Art und Weise auch im inneren Ohr des Menschen abläuft [Roe79]. Die andere Sichtweise bezieht sich auf Eigenschaften der menschlichen Wahrnehmung wie Lautstärke oder Harmonieempfinden. Ausgehend von psychoakustischen Untersuchungen wurden Modelle der menschlichen Wahrnehmung von Musik und Sprache entwickelt. Diese Modelle wurden auf den Computer übertragen, um das menschliche Hörempfinden zu simulieren. Die Ausgangswerte diese Modelle können ebenfalls zu einem Eigenschaftsvektor kombiniert werden, der ein Musikstück beschreibt. Eine detai-

lierte Beschreibung von Möglichkeiten der Audiodatenanalyse ist in [PFE96] zu finden.

1.2 Ähnlichkeitssuche in Datenbanken

Für Ähnlichkeitssuche in Datenbanken werden in der Regel nur die Eigenschaftsvektoren verwendet. Im Datenbankbereich wurden verschiedene Indexstrukturen beschrieben, welche Nächsten-Nachbarnsuche auf mehrdimensionalen Vektoren unterstützen. Ein Überblick über den aktuellen Forschungsstand ist in [BBK01] zu finden. Neben dem dort behandelten Problem der Effizienzsteigerung der Nächsten-Nachbarnsuche wurde in den letzten Jahren auch das Effektivitätsproblem behandelt. Hier wird untersucht, wie aussagekräftig die Ergebnisse einer Anfrage auf hochdimensionalen Daten sind. Die Fragestellung wurde in [BGRS99, AHK01] theoretisch untersucht und für verschiedene Datenverteilungen gezeigt, daß mit steigender Dimensionalität der Abstand von einem beliebigen Anfragepunkt zum nächsten Nachbarn schneller steigt als die Differenz zwischen den Abständen zum nächsten bzw. zum weitesten Nachbarn, d.h. daß der Kontrast zwischen dem nächsten und weitesten Nachbarn mit steigender Dimensionalität abnimmt. In [HAK00] wurde vorgeschlagen, statt im hochdimensionalen Datenraum in Unterräumen mit niedrigerer Dimensionalität nach aussagekräftigen nächsten Nachbarn zu suchen. Das Problem der Nächsten-Nachbarnsuche in einer Projektion des Datenraumes ist viel komplexer als eine einfache Nächsten-Nachbarnanfrage, da die Anzahl der potentiell interessanten Projektionen sehr groß ist. In [HAK00] wurde das Problem mittels eines genetischen Algorithmus gelöst, dessen Fitnessfunktion bewertet, wie stark die Datenpunkte um den Anfragepunkt geclustert sind. In [HK00] wurde eine allgemeine interaktive Variante diese Methode vorgeschlagen. Anstelle der automatischen Fitnessfunktion entscheidet der Benutzer interaktiv, welche der vom System vorgeschlagenen Projektionen eine aussagekräftige Nächsten-Nachbarnsuche erlaubt. Innerhalb weniger Iterationen paßt sich das System dem Ähnlichkeitsbegriff des Benutzers an, soweit die notwendige Information in den Eigenschaftsvektoren kodiert ist. Wichtig für diese Art der Ähnlichkeitssuche ist eine geeignete Visualisierung für die Objekte. In [HK00] wurden unter anderem Anwendungen für interaktive Bildähnlichkeitssuche untersucht, bei denen eine Visualisierung schon in natürlicher Art und Weise durch die jeweiligen Bilder gegeben ist.

1.3 Überblick

In diesem Artikel wird eine Methode zur interaktiven, visuellen Musikähnlichkeitssuche beschrieben. In Kapitel 2 werden Grundlagen zur Musikverarbeitung eingeführt. In Kapitel 3 wird ein einfaches Verfahren zur Berechnung von zeitunabhängigen Eigenschaftsvektoren aus beliebig langen Musikstücken entwickelt und darauf aufbauend eine neue Visualisierungstechnik vorgestellt, die es erlaubt, beliebig lange Musikstücke visuell miteinander zu vergleichen. Beide Komponenten werden in *MusicOpt*, einem datenbankgestütztem, interaktiven Retrieval-System, integriert. Abschließend wird in Kapitel 4 die neue interaktive Methode mit einem bekannten Musik-Retrieval-Verfahren verglichen.

2 Grundlagen der Musikverarbeitung

Zur einfachen Charakterisierung der Audiodaten kann die Fourier-Transformation herangezogen werden. Der Grundgedanke der Fourier-Transformation ist, daß beliebig komplexe Signale aus einer Summe von Sinusschwingungen unterschiedlicher Frequenz, Amplitude und Phase zusammengesetzt werden können. Die Aufgabe der Signalanalyse ist also die Zerlegung eines Signals in seine einzelnen harmonischen sinusförmigen Teilschwingungen.

Die Audiodaten liegen grundsätzlich als kontinuierliche Signale vor, die zur rechnergestützten Bearbeitung erst digitalisiert werden müssen. Für die Digitalisierung – auch Sampling genannt – wird das analoge Signal zu diskreten Zeitpunkten abgetastet und per Analog / Digital-Wandler in diskrete Werte umgewandelt. Auf der Zeitachse spricht man von Abtastrate(-frequenz), auf der Amplitudenachse von Quantisierung. Die Abtastfrequenz sollte nach dem Nyquist-Theorem bestimmt werden, d.h die Abstastfrequenz muß mindestens doppelt so groß sein, wie die größte im Signal vorkommende Frequenz. Dies stellt sicher, daß das analoge Signal aus der digitalen Form wieder zurückgewonnen werden kann. Da das menschliche Gehör Frequenzen von 20Hz bis 20kHz wahrnehmen kann, werden Audiodaten in der Regel mit einer Abtastfrequenz von 44,1kHz gesampelt.

Die Quantisierung hat auch großen Einfluß auf die Klangqualität des digitalen Signals. Der Amplitudenbereich wird in eine Anzahl von Intervallen unterteilt, so daß für jeden Abtastwert nur der Index des betroffenen Bereichs gespeichert werden muß.

Das Resultat der Digitalisierung ist ein zeitdiskretes Signal s, auf das die diskrete Fourier-Transformation angewendet werden kann. Falls $s(n)$ für $n \in \{0, 1, 2, \ldots, N-1\}$ definiert ist, so ist die N-Punkt diskrete Fourier-Transformation (N-Punkt DFT) definiert als:

$$S(f) = \sum_{n=0}^{N-1} s(n) e^{-j\frac{2\pi}{N} \cdot n \cdot f} \quad \texttt{für} \quad f = 0, 1, \ldots, N-1 \quad \texttt{und} \quad j = \sqrt{-1} \tag{1}$$

Die schnelle Fourier-Transformation (Fast-Fourier-Transformation / FFT) ist ein komplexer Algorithmus, der die Berechnungskomplexität von $O(n^2)$ auf $O(n \cdot \log n)$ verringert [Bri74].

Da für die weitere Verarbeitung eine Fourier-Transformation eines Musikstückes als Ganzes nur die durchschnittliche Frequenzverteilung liefert und die FFT des gesamten Musikstückes sehr viel Rechenzeit in Anspruch nimmt, läuft die Frequenzzerlegung der Audiodaten folgendermaßen ab. Das Audiosignal wird in gleich große, sich überlappende Fenster (typischerweise 16 ms) zerlegt. Auf jedem Fenster wird eine diskrete Fourier-Transformation (N-Punkt DFT) durchgeführt, um das Signal in seine Frequenzanteile zu zerlegen.

Da es durch ein diskretes Abschneiden des Signals an den Rändern des betrachteten Fensters zu Verfälschungen kommen kann, wird das Signal an den Rändern durch eine sogenannte Fensterfunktion ein- und ausgeblendet. Zur Fensterbildung wird in dieser Arbeit die Hamming-Funktion verwendet [Sch00].

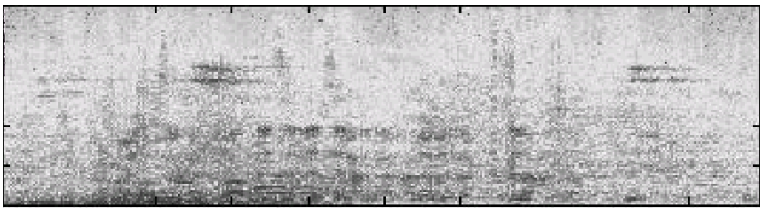


Abbildung 1: Die Abbildung zeigt eine 2-dimensionale zeitabhängige Frequenzintensitätsfunktion, bei der die Zeit auf der x-Achse und die Frequenz auf der y-Achse liegen. Der Intensitätswert ist als Grauwert dargestellt (hell entspricht geringer, dunkel hoher Intensität).

Durch die Fensterunterteilung entsteht als Resultat der Fourier-Transformationen eine zweidimensionale Frequenzintensitätsfunktion, die für jedes Zeitintervall (Fenster) ein Frequenzspektrum liefert. Abbildung 1 zeigt ein Beispiel für eine zeitabhängige Frequenzintensitätsfunktion. Die Grauwerte der Pixel zeigen, mit welcher Intensität die jeweiligen Frequenzen auftreten.

3 Interaktive visuelle Musikähnlichkeitssuche

In diesem Kapitel wird ein einfaches Verfahren zur Berechnung von zeitunabhängigen Eigenschaftsvektoren aus beliebig langen Musikstücken entwickelt und darauf aufbauend eine neue Visualisierungstechnik vorgestellt, die es erlaubt, beliebig lange Musikstücke visuell miteinander zu vergleichen. Beide Komponenten werden in *MusicOpt*, einem Datenbank gestütztem, interaktiven Retrieval-System integriert.

3.1 Eine effiziente Musikähnlichkeitsmetrik

Die FFT der überlappenden Zeitfenster liefert ein zeitliche Reihe von Frequenzspektren mit Intensitätswerten. Diese sehr genaue Darstellung ist für effiziente Ähnlichkeitsvergleiche aufgrund ihrer Größe ungeeignet. Für die Ähnlichkeitssuche ist eine kompakte, zeitunabhängige Darstellung sinnvoll. Um eine Beschreibung mit den erforderlichen Eigenschaften zu erhalten, wird ein Histogramm berechnet, in welchem gezählt wird, wie oft die jeweiligen Frequenz-Intensitätskombinationen in dem Musikstück auftreten. In Abbildung 2 wird ein Überblick über diese neue Transformation dargestellt. Die Transformation wird im folgenden etwas genauer beleuchtet.

Wie in Kapitel 2 beschrieben, stellt die Frequenzintensitätsfunktion eine sehr genaue, zeitabhängige Beschreibung des Audiosignals dar. Für jedes Zeitfenster wird eine zweidimensionale Ausgabe bestehend aus Frequenzbereich und Intensität erzeugt. Die Größen Frequenz und Intensität werden in Abbildung 1 auf y-Achse und Grauwert gelegt. Um ein kompakteres und zeitunabhängiges Histogramm zu erhalten, werden die zeitlich verschiedenen 2D-Ausgaben der FFT für die Zeitfenster erstens in grobere Bereiche unterteilt und

zweitens über die Zeit aggregiert.

Für die Vergröberung wird der Frequenzbereich von 20Hz-22kHz logarithmisch in k Frequenzbereiche eingeteilt. Die logarithmische Unterteilung wurde gewählt, da der Mensch die Frequenzen nahezu logarithmisch und nicht linear wahrnimmt [Roe79]. Die Intensitäten werden in Dezibel umgerechnet und der Intensitätsbereich wird ebenfalls in l Bereiche unterteilt. In den beschriebenen Anwendungen erstreckt sich der genutzte Intensitätsbereich von 0-60 Dezibel. Das Histogramm besteht also aus k Frequenzbereichen mit jeweils l Intensitätsbereichen. Typische Werte für die Anzahl der Frequenz- bzw. Intensitätsbereiche sind $k = 40$ und $l = 20$. Anschließend wird die Frequenzintensitätsfunktion nach der Zeit durchlaufen, wobei die Häufigkeiten der auftretenden Frequenzintensitätskombinationen gezählt werden.

Diese Frequenz-Histogramme können als Grundlage einer effizienten Vergleichsmetrik für Musikstücke dienen. Zwei Frequenz-Histogramme a, b werden mittels euklidischer Metrik wie folgt verglichen:

$$dist(a,b) = \sqrt{\sum_{i=1}^{k}\sum_{j=1}^{l}(a_{i,j} - b_{i,j})^2} \tag{2}$$

Diese Metrik kann für die Nächsten-Nachbarnsuche verwendet werden. Eine einfache Verallgemeinerung auf Nächsten-Nachbarnsuche in Projektionen [HAK00] ist zwar möglich, aber die genetische Suche bleibt wegen der hohen Dimensionalität ($d = 20 \cdot 40 = 800$) uneffektiv. Eine sinnvolle Einschränkung des Projektionssuchraumes wird durch die Forderung erreicht, daß alle Intensitätsbereiche derselben Frequenz immer vollständig in einer Projektion enthalten sein müssen.

3.2 Musik-Visualisierung

In diesem Abschnitt wird ein Verfahren beschrieben, wie aus den ermittelten Frequenz-Histogrammen der Musikstücke Visualisierungen generiert werden können. Diese werden für das interaktive genetische Verfahren benötigt, bei dem der Benutzer entscheidet, welche Projektion eine relevante Ähnlichkeitssuche erlaubt. Die in [HAK00] vorgeschlagene automatische Fitnessbewertung ist hier nicht sinnvoll anwendbar, da oft nur sehr wenige relevante Ergebnisse in einer Datenmenge vorkommen können. Somit ist die für die automatische Fitnessfunktion wichtige Annahme nicht mehr gültig, daß die relevanten Daten um den Anfragepunkt geclustert sind.

Um die zwei-dimensionalen Frequenz-Histogramme platzsparend darzustellen, wurden die Häufigkeitswerte auf Graustufen abgebildet. Jedoch eine einfache Abbildung reichte bei diesen Daten nicht aus. Da wenige Histogrammzellen sehr stark belegt sind, ergeben die Histogramme nach der Abbildung auf Grauwerte ein kaum strukturierte Visualisierung (Abb. 3(a)). Um den Kontrast zu erhöhen wurde in den weiteren Teilabbildungen (b-d) nicht die Häufigkeit h sondern $h^{1/2}$, $h^{1/4}$ bzw. $h^{1/8}$ dargestellt. Abbildung 3(c) weist den besten visuellen Kontrast auf und daher wurde $h^{1/4}$ für die Visualisierungen verwendet.

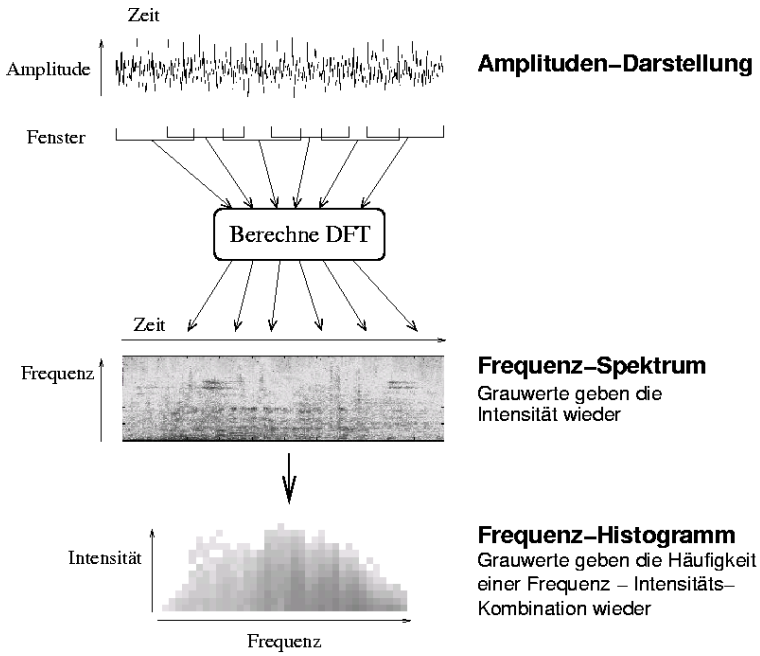


Abbildung 2: Die Abbildung zeigt eine Übersichtsskizze der Prozedur zur Transformation der Audiodaten in eine zeitunabhänige Vektordarstellung. Das Ergebnis der Umwandlung ist ein Histogramm, das die Häufigkeiten von Frequenzintensitätskombinationen beschreibt. Das Histogramm entsteht durch Zählung der auftretenden Kombinationen über die Dauer des Musikstückes (oder eines Teils des Stückes).

Die Kontrastqualität wurde noch auf weiteren Beispielen geprüft und immer ergab $h^{1/4}$ den besten visuellen Eindruck. Die Suche nach einer befriedigenden Erklärung für diesen Sachverhalt ist Gegenstand weiterer Forschungen.

3.3 Das MusicOpt-System

In dem Musik-Retrieval-System *MusicOpt* wurde das angepaßte interaktive Verfahren zur Ähnlichkeitssuche in Unterräumen mit den Musikvisualisierungen kombiniert. Mittels interaktiver Auswahl der relevanten Projektionen kann der Benutzer dem System sein Ähnlichkeitsmaß mitteilen, ohne den Transformationsprozess der Musikdaten zu den Frequenz-Histogrammen im Detail verstehen zu müssen. Dieser interaktive Prozess führt in den meisten Fällen zu besseren Suchergebnissen.

Abbildung 5 zeigt ein Bildschirmfoto des *MusicOpt*-Systems. Ein Zeile der Bildmatrix zeigt jeweils das Ergebnis einer Nächsten-Nachbarnsuche in einem bestimmen Unterraum. In der ersten farbig hinterlegten Zeile ist das Ergebnis der Nächsten-Nachbarnsuche, bei der alle Dimensionen berücksichtigt werden, dargestellt. Bei den übrigen sind die Fre-

(a) (b) (c) (d)

Abbildung 3: Die Abbildungen (a-d) zeigen das selbe Frequenz-Histogramm mit unterschiedlichen Häufigkeitstransformationen. Für Abbildung (a) wurde die Häufigkeit linear auf die Grauwerte abgebildet, bei den Abbildungen (b-d) wurde $h^{1/2}$, $h^{1/4}$ bzw. $h^{1/8}$ verwendet. Ein visueller Vergleich ergibt, daß für $h^{1/4}$ die Grauwertskala am besten ausgeschöpft wurde.

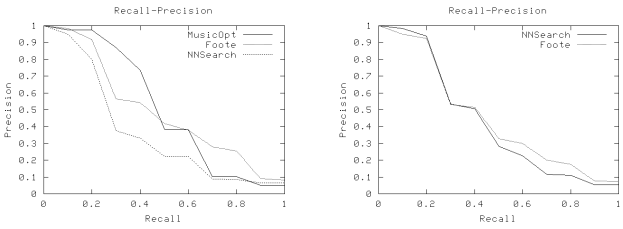
quenzbereiche, die den genutzten Unterraum definieren, mit einem schwarzen Punkt markiert. Um die Relevanz der Ergebnisse zu beurteilen, kann der untrainierte Benutzer zuerst einige Stücke akustisch durchhören, um so Musik und Visualisierung miteinander in Verbindung zu bringen. Später kann der Benutzer die für ihn relevanten Ergebnisse auf visueller Basis auswählen. Das System erzeugt mittels einfacher Cross-Over Operatoren neue Varianten der gewählten Projektionen. Falls das Ähnlichkeitsverständnis des Benutzers mittels einer Projektion der zugrundeliegenden Frequenz-Histogramme ausdrückbar ist, konvergiert das System in der Regel nach drei bis vier Iterationen.

4 Evaluierung

In diesem Abschnitt wird der interaktive Ansatz zur Musikähnlichkeitssuche mit dem Verfahren von Foote [Foo97] verglichen. Zum diesem Ansatz war das Suchsystem und die Datenbasis unter `http://www.fxpal.com/people/foote/musicr/doc0.html` verfügbar. Die Datenbasis besteht aus 255 Musikstücken, die auf jeweils 7 Sekunden gekürzt wurden. Die Musikstücke stammen von 40 verschiedenen Bands und Interpreten. Die Ergebnisse des Ansatzes von Foote sind vorberechnet in den Webseiten gespeichert. Eine wichtige Frage bei der Evaluierung ist die Definition der relevanten Ergebnisse zu einer Anfrage. Für diese Arbeit wurde die oft genutzte Annahme verwendet, daß zu einem Anfragestück die Stücke der gleichen Musikgruppe relevant sind. Für die Auswertung wurden die üblichen Maße wie Precision und Recall genutzt, ebenso wie die Standard-Auswertungssoftware der Information-Retrieval Konferenz TREC `http://trec.nist.gov/`. Precision und Recall sind wie folgt definiert:

$$Precision = \frac{Number\_Retrieved\_Relevant}{Number\_Total\_Retieved}, Recall = \frac{Number\_Retrieved\_Relevant}{Number\_Possible\_Relevant} \quad (3)$$

Ziel eines Suchsystem ist es, daß beide Werte möglichst groß sind. Die Abbildung 4(a) zeigt die durchschnittliche Recall-Precision-Kurve über 10 Anfragen. Es ist zu sehen, daß mit dem interaktiven *MusicOpt*-Ansatz mit höherer Genauigkeit deutlich mehr relevante Treffer erzielt werden können als mit dem automatischen Ansatz von Foote. Besonders wichtig ist der folgende Bereich: Recall[0.2:0.5], Precision[0.6:1], weil mit einer Einstellung aus diesem Bereich die Ergebnisse dem Benutzer zuerst präsentiert werden. Da entgegen der Annahme, daß alle Musikstücke einer Gruppe relevant sind, bei den meisten Gruppen starke Inhomogenitäten in ihren Stücken bestehen, spiegelt ein Recall größer als 0.5 oft nicht die gewünschten Ergebnisse wieder. In Abbildung 4(b) verglichen wir die



(a) (b)

Abbildung 4: Teil (a) Vergleich Foote, Nearest-Neighbour-Search und MusicOpt für 10 Anfragen; Teil (b) Vergleich Foote, Nearest-Neighbour-Search für 40 Anfragen, (Werte rechts oben sind besser)

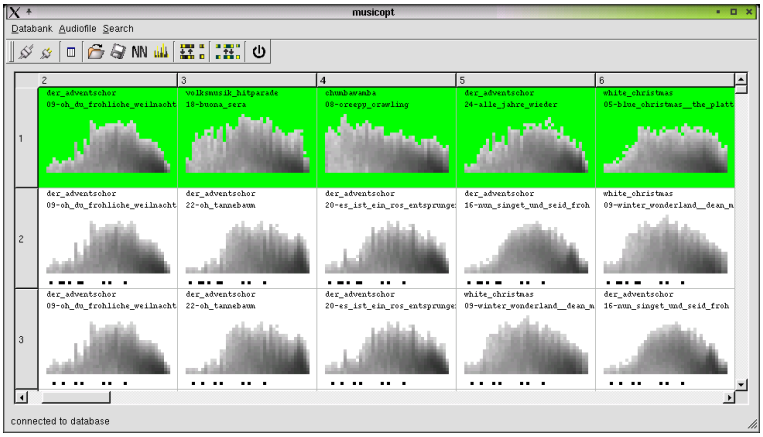


Abbildung 5: Die Abbildung zeigt ein Bildschirmfoto des *MusicOpt*-Systems.

Standard-Nächste-Nachbarnsuche mit den Ergebnissen von Foote auf 40 Anfragen. Beide Verfahren arbeiten etwa gleich gut, so daß der Effektivitätsgewinn in Abbildung 4(a) allein der interaktiven *MusicOpt*-Methode zuzurechnen ist.

In Abbildung 5 ist das Potential der Effektivitätssteigerung an einem Beispiel gezeigt. Als Anfragepunkt würde ein Lied eines Adventschors gewählt. Die Standard-Nächste-Nachbarnsuche liefert nur ein weiteres Lied dieses Chors zurück. Mit Hilfe der *MusicOpt*-Methode läßt sich diese Anzahl auf drei erhöhen, wobei nicht-relevante Stücke nicht mehr im Ergebnis auftauchen. Die verbesserte Qualität läßt sich auch anhand der Visualisierungen verifizieren.

In weiteren Forschungen sollen die Möglichkeiten der Rhythmuserkennung und abstrakte, allegorische Visualisierungen zur Verbesserung der Musikähnlichkeitssuche untersucht werden.

Literaturverzeichnis

[AHK01] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *ICDT 2001 (International*

Conference on Database Theory), London, UK, 2001. Springer, 2001.

[BB01] Jerome Barthélemy and Alain Bonardi. Figured bass and tonality recognition. In *Proceedings of the Second Annual International Symposium on Music Information Retrieval: ISMIR 2001*, pages 129–136. Indiana University, 2001.

[BBK01] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)*, 33(3):322–373, 2001.

[BGRS99] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When Is "Nearest Neighbor" Meaningful? In *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer, 1999.

[Bri74] E. O. Brigham. *The Fast Fourier Transform*. Prentice-Hall Inc., 1974.

[DC01] Adriane Swalm Durey and Mark A. Clements. Melody spotting using hidden Markov models. In *Proceedings of the Second Annual International Symposium on Music Information Retrieval: ISMIR 2001*, pages 109–117. Indiana University, 2001.

[Foo97] J. Foote. Content-based retrieval of music and audio. In *Multimedia Storage and Archiving Systems II, Proceedings of SPIE*, pages 138–147, 1997.

[HAK00] Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. What Is the Nearest Neighbor in High Dimensional Spaces? In *VLDB'2000, Proceedings of 26th International Conference on Very Large Data Bases, Cairo, Egypt*. Morgan Kaufmann, 2000.

[HBABS00] Perfecto Herrera-Boyer, Xavier Amatriain, Eloi Batlle, and Xavier Serra. Towards instrument segmentation for music content description: a critical review of instrument classification techniques. In *Proceedings of the Second Annual International Symposium on Music Information Retrieval: ISMIR 2000*. University of Massachusetts at Amherst, 2000.

[HK00] Alexander Hinneburg and Daniel A. Keim. Using Visual Interaction to solve Complex Optimization Problems. In *Dagstuhl seminar on Scientific Visualization*. Kluwer, May 2000.

[KNS+00] Naoko Kosugi, Yuichi Nishihara, Tetsuo Sakata, Masashi Yamamuro, and Kazuhiko Kushima. A practical query-by-humming system for a large music database. In *Proceedings of the eighth ACM international conference on Multimedia*, pages 333–342. ACM Press, 2000.

[PFE96] Silvia Pfeiffer, Stephan Fischer, and Wolfgang Effelsberg. Automatic audio content analysis. In *Proceedings of the fourth ACM international conference on Multimedia*, pages 21–30. ACM Press, 1996.

[PRM02] E. Pampalk, A. Rauber, and D. Merkl. Content-based Organization and Visualization of Music Archives. In *Proceedings of ACM Multimedia 2002*, France, 2002. ACM.

[Rap01] Christopher Raphael. Automated rhythm transcription. In *Proceedings of the Second Annual International Symposium on Music Information Retrieval: ISMIR 2001*, pages 99–107. Indiana University, 2001.

[Roe79] J. G. Roederer. *Introduction to the Physics and Psychophysics of Music*. Springer, New York, 1979.

[Sch00] Carsten Schäfer. Entwicklung einer Audio-Retrieval-Komponente für ein multimediales IR-System. Master's thesis, University Dortmund, 2000.

[WBKW96] Erling Wold, Thom Blum, Douglas Keislar, and James Wheaton. Content-Based Classification, Search, and Retrieval of Audio. *IEEE MultiMedia*, 3(3):27–36, 1996.

[YK99] Chi Lap Yip and Ben Kao. A Study of Musical Features for Melody Databases. In *Proceedings l0th International Conference on Database and Expert Systems Applications*, pages 724–733, 1999.

An Ontology for Domain-oriented Semantic Similarity Search on XML Data

Anja Theobald
Department of Computer Science
University of the Saarland, Germany
WWW: http://www-dbs.cs.uni-sb.de
E-mail: theobald@cs.uni-sb.de

Abstract: Query languages for XML such as XPath or XQuery support Boolean retrieval where a query result is a (possibly restructured) subset of XML elements or entire documents that satisfy the search conditions of the query. Web search engines, on the other hand, are based on the ranked retrieval paradigm, but do not consider the additional information and rich annotations provided by the structure of XML documents and their element names. Furthermore, web search engines have very little "semantic" data and are thus unable to cope with ambiguous search terms. Ontological knowledge and appropriate index structures are necessary for semantic similarity search on XML data extracted from the web. In this paper we present a powerful ontology index which supports domain-specific semantic similarity search with a new measure for expressing the relevance of query results.

1. Introduction

1.1 Motivation

XML is becoming the standard for integrating and exchanging data over the Internet and within intranets, covering the complete spectrum from largely unstructured, ad hoc documents to highly structured, schematic data. For searching information in open environments such as the Web or intranets of large corporations, ranked retrieval is required: a query result is a rank list of XML elements in descending order of relevance.
If you are looking for information about the composition of a chip you will get many relevant documents. But this is a mix of information about games (e.g. poker), hardware (e.g. microchip), animals (e.g. cow chip), food (e.g. potato chip) and further domains. The ambiguity of words and the need for a semantically meaningful interpretation of data call for ontology-based domain-oriented semantic similarity search and an appropriate ontology index.

Definition 1: An *ontology/thesaurus* is a specification of representational vocabulary of words/*terms* including *hierarchical relationships* and *associative relationships* between these words. It is used for indexing and investigation as well as for support knowledge sharing and reuse [Gru93, Gua98].

Consider the word "chip". For this term we know several related terms:

snack food	— is a broader term (hypernym) of →	chip
microprocessor	— is a narrower term (hyponym) of →	chip
blue chip	— is a narrower term (hyponym) of →	chip

If we insert this information into a simple tree-based term hierarchy (see Figure 1a) as described in [STW01, TW02] we obtain the wrong impression that "snack food" and "blue chip" are related terms.

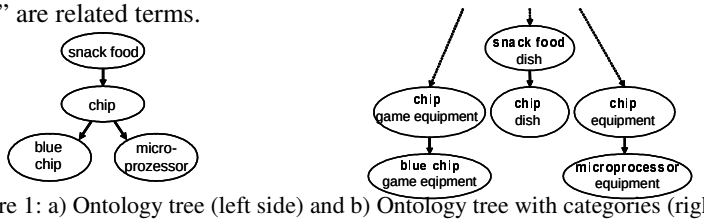


Figure 1: a) Ontology tree (left side) and b) Ontology tree with categories (right side)

So we need a method to distinguish different senses of a term. For this purpose we introduce a notion of a *category* for more precisely specifying the meaning of a term. As a representative of a category for term t we use a broader term (hypernym) that comprises t as a narrower concept (hyponym), e.g. term: **chip** – category: microchip. The upper bold-faced entries are ontology terms and the second entries are the corresponding categories (see also Figure 1b). To produce a ranked result for an ontology-based search we need a similarity assessment of a relationship between two ontology terms.

For elaborating the idea above an obvious approach would be to use an existing thesaurus. For example, WordNet [WN98] is an extensive electronic lexical database. For a given search term WordNet is able to distinguish several senses of this term and is able to return related terms to the given term. Unfortunately, WordNet supports only term-based search (no search for a given sense) without any quantification of the relationships between related terms.

Our work extracts semantic information from WordNet (and possibly other ontological sources) for building a quantified graph-like structure as a backbone for XML similarity search.

1.2 Related Work

Researchers in artificial intelligence first developed logic-based ontologies to facilitate knowledge sharing and reuse. In the last decade ontologies have become a popular research topic and several AI research communities such as knowledge engineering, natural language processing and knowledge representation have investigated them. The interest in ontologies has been revived with the recent discussion about the "Semantic Web" [SemWeb, MWK00, SAD+00]. In contrast to the extremely ambitious early AI approaches toward building universal ontologies (see, e.g., [LG90, RN95]), more recent proposals are aiming at domain- or user-specific ontologies and are based on more tractable logics (see, e.g., [Hor02, SAD+00]).

Recent publications on formalization of ontologies cover a wide spectrum from algebraic approaches [BM99] and logic-based languages for modeling ontologies [DEFS99, BEH+02] to ontologies for conceptual data modeling and data interpretation [Gua98, NFM00, SM00]. Similar work has been done in the context of multi-databases [BHP94, PM98]. These publications do not consider quantification of relationships. Only few papers [KKS01, KKC94, KI90, STW01, BHP94] describe weighted ontologies. These approaches often define simple weights for term similarity not appropriate for query processing.

To our knowledge, the role of ontologies in searching semistructured data has not yet been discussed in any depth. The unique characteristic of our approach lies in the combination of ontological knowledge and information retrieval techniques for domain-oriented semantic similarity search on XML data.

1.3 Contribution and Outline of the Paper

In this paper we present a combination of ontological knowledge and information retrieval techniques for domain-oriented semantic similarity search on XML data. For this purpose we have developed an ontology which contains a category for each term to recognize domains. Furthermore, we introduce a quantification of relationships between ontology terms using similarity weights. For automatic ontology construction we build on the WordNet thesaurus. For semantic domain-oriented querying XML data we use the flexible XML search language called XXL.

The rest of the paper is organized as follows. Section 2 gives a short description of the flexible XML search language called XXL and the architecture of the XXL search engine. Section 3 presents the modeling of the ontology index using categories and differentiated relationships between ontology terms. Section 4 outlines the application of the ontology index for evaluation of semantic similarity search conditions within XXL queries.

2. XXL: A Flexible XML Search Language for Ranked Retrieval

For the flexible XML search language called XXL we have adopted several concepts from XML-QL, XQuery and similar languages as the core, with certain simplifications and resulting restrictions.

For searching publications over chips according to the example scenario given in the motivation we can express such a query in our language XXL as follows:

```
SELECT   T                          // output of the XXL query
FROM     INDEX                      // search space of the XXL query
WHERE    #.~publication AS P        // search condition
   AND   P.title AS T
   AND   P.description ~ "chip"
```

We define the Where clause of a query as the logical conjunction of *path expressions*, where a path expression is a regular expression over *elementary conditions* and an elementary condition refers to the name or content of a single element or attribute.

In contrast to other XML query languages we introduce a new operator "~" to express semantic similarity search conditions on XML element names as well as on XML element contents. The result of an XXL query is a subgraph of the XML data graph, where the nodes are annotated with local relevance probabilities called *similarity score* for the elementary search conditions given by the query. These similarity scores are combined into a global similarity score for expressing the relevance of the entire result graph. Full details of the semantics of XXL and especially the probabilistic computation of similarity scores can be found in [TW00, TW02].

This XML search language is implemented within the XXL search engine. The XXL search engine is a client-server application with a Java-based GUI. The server programs

consist of 1) service components: the crawler and the query processor (both Java servlets), 2) algorithmic components: parsing and indexing documents, parsing and checking XXL queries, and 3) data components: data structures and their methods for storing various kinds of information like the element path index (EPI), the element content index (ECI), and the element-name ontology index (NOI). For storing the ECI we use Oracle.

An XXL query is evaluated as follows [TW02]. The Where clause of an XXL query is a logical conjunction of n regular expressions for search conditions over XML element paths. The query processor (QP) decomposes the given query into n subqueries and constructs a graph-based representation for each subquery similar to a finite state automaton. The global order of evaluating the n subqueries and the local evaluation strategy (e.g., top-down vs. bottom-up) for each subquery are automatically chosen by the QP.

For each subquery, simple path expressions with element names and the wildcard symbol # are looked up in the EPI. For example, all occurrences of a pattern *#.publication.description* can be retrieved from the EPI. Content conditions are evaluated by the ECI, a text index on element and attribute contents. For "semantic" similarity conditions such as *description ~ "chip"* the ECI yields approximate matches and a similarity score based on IR-style tf*idf measures [BR99] and "semantic distances" between concepts in the ontology. Finally, for semantic similarity conditions on element names, for example, *~publication*, the NOI is used to expand the query in order to capture semantically related element names such as *article*; again, a similarity score is computed for result ranking.

The ranked lists of relevant subgraphs from the index-based subquery evaluation are composed into a list of global graphs, each of which has assigned to it a global similarity score derived from the local scores by elementary probability computation. The QP finally extracts the result as specified by the Select clause and constructs an XML document that is returned to the XXL client.

3. Category-based Ontology

Recall from Definition 1 that an *ontology* is a specification of representational vocabulary of *terms* and contains *relationships* between these terms. Now we present a definition of an ontology containing terms and their categories and relationships between terms based on an existing thesaurus which is able to produce related terms to a given term. In our approach we will use WordNet as a common sense thesaurus.

3.1 Graph-based Data Structure

Our approach pursues the several goals, e.g. modelling of element names and keywords of element contents in separate ontology indexes, improving the similarity weights for expressing correlations between ontology terms, construct the ontology index automatically, and implementing a tool for human management of an ontology index.
For this purpose we have developed an ontology index which supports domain-oriented search using categories as well as more detailed relevance ranking using several relationships between two ontology terms.

First we give a short description of the most important features of WordNet necessary for our following work. For a search term WordNet provides synonyms, 1-level hy-

pernyms (broader terms), 2-level hypernyms, … , hyponyms (narrower terms), meronyms (part of something), holonyms (whole of something) for each meaning of the given term. A 2-level hypernym is more general than a 1-level hypernym, etc.

Definition 2: A *category-based ontology index* (OI) is a directed, labeled graph G=(V,E) where V is a finite set of nodes and E is a finite set of edges. Each node n=(t,c) consists of an ontology *term* t and its *category* c.

For the *category* c of an ontology term t we choose the 2-level hypernym provided by WordNet which best describes the meaning of t. The ontology index contains a special root node n=(entity,entity) where entity is called *top term*. This node has only outgoing edges.

There is an edge between two nodes n1 and n2, if n1.t and n2.t are related terms (e.g. n1.t is a hypernym of n2.t). An edge between two nodes n1 and n2 is a tuple e=(n1,n2,type,weight) where type ∈ {is_synonym_of, is_hypernym_of, is_hyponym_of, is_holonym_of, is_meronym_of} defines the relationship between the two terms n1.t and n2.t and its corresponding categories n1.c and n2.c and the weight denotes the similarity of the ontology terms of n1 and n2 according to its categories.

Figure 2 displays a part of such an ontology index where a node contains the bold faced ontology term and its category. Each edge is labeled with a relationship and a weight between two connected ontology terms (hyper is a shortcut for "is_hypernym_of", etc.). The given weights are fictitious. It is not necessary to model the edges of type hyponym and meronym explicitly, because they are reverse directions of hypernym and holonym, respectively. The relationship "Synonym" is a symmetric, reflexive, and transitive relation.

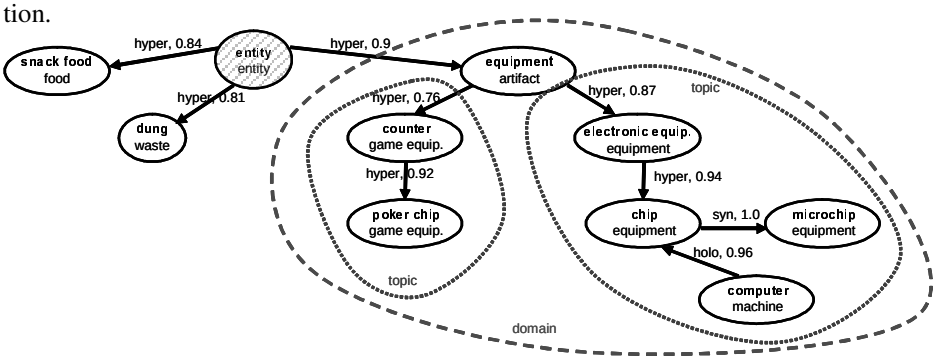


Figure 2: Graph-based ontology index

The ontology index represents the relationships of terms as well as the relationships of categories.

Definition 3: A *topic* is a subgraph of OI which consists of nodes with the same category. All topics are distinct subgraphs in terms of their set of nodes. Two topics are connected, if there exists at least one edge between a node of the first topic and a node of the second topic. Each category of a child of the root=(entity,entity) of the OI is a root (*domain node*) of a subgraph called *domain*.

According to Definition 2 each edge is also denoted with a *weight* to express the similarity of the two corresponding nodes. For a given path p we define the similarity between the start node and the end node using all weights of the corresponding edges and their

distance to the start node. To obtain a fair result we consider the forward path and the backward path of the given path p as defined in Definition 4.

Definition 4: The *similarity sim(v_i,v_j)* between two terms of two ontology nodes v_i and v_j connected by a path p_{ij}=<v_i …v_j> is computed as follows.
Let p_{ij}=<n_0…n_k> where v_i=n_1 and v_j=n_k the forward path and pji=<n_k…n_0> the backward path of p between nodes v_i and v_j. Let length(p)=|p_{ij}|=|p_{ji}|=k.

$$sim(v_i,v_j) = \frac{(weight(p_{ij}) + weight(p_{ji}))}{2 \cdot length(p)}$$

$$weight(p_{ij}) = \sum_{m=0}^{length(p)-1} \frac{length(p)-m}{length(p)} weight(< n_m, n_{m+1} >)$$

$$weight(p_{ji}) = \sum_{m=0}^{length(p)-1} \frac{length(p)-m}{length(p)} weight(< n_{m+1}, n_m >)$$

For example, let v.t=”electronic equipment” be a given search term and w.t=”microchip” a similar term as shown in Figure 2. The similarity score sim(v,w) = ((1·0.94 + 1/2·1.0) + (1·1.0 + 1/2·0.94)) / (2·2) = 0.73.

The rationale for this formula is that the length of a path has direct influence on the similarity score. That means, the similarity score for a short path will be better than for a longer one. We need an appropriate algorithm for computing shortest path with best similarity.

3.2 Automatic Building of Ontology Indexes

For building an ontology index from scratch we start with a top down method (specialization) according to the given top term “entity”. For the subsequent insertion of nodes and for the corresponding update process we use a hybrid method. That means, we determine a set of related terms to a given term and uses specialization method (top down) as well as generalization method (bottom-up) for insertion all nodes.

The following algorithm consists of seven main steps. At the beginning we have an ontology index with one node n=(entity,entity) for the top term “entity”. For illustration consider the given term “chip” and the ontology index as shown in Figure 2.

Step 1: Extracting a term from an XML document during parsing this document.
During parsing of an XML document an appropriate term (noun) will be extracted. Each element name and each important keyword of an element content (e.g., each noun) is an appropriate term for insertion into the OI. For extracting keywords from element content we are using stopword elimination and statistical information about term frequencies.
Step 2: Finding categories and related terms to the term given by step 1 with the help of an existing thesaurus like WordNet.
For the term “chip” we obtain several meanings. For each meaning we obtain following categories according to the 2-level hypernyms provided by WordNet for the given term according to each sense:

i) fecal matter ii) dish iii) game equipment iv) equipment

This way we have four *basic nodes*, e.g n1=(chip, fecal matter). Now we are able to compute *related nodes* for each basic node using WordNet. In our example we obtain a set of related terms for the given term “chip”, e.g.:

	i) fecal matter	ii) dish	iii) game equipment	iv) equipment
synonyms:	cow chip,…	potato chip,…	poker chip,…	microchip,…
hypernyms:	dung,…	snack food,...	counter,...	elect. equip.,...
…				

Step 3: Computing the weights for two given terms using standard web search engines like Google, Yahoo, Altavista, etc.

For a given term and one of its related terms we use a standard web search engine to get a weight for the similarity of these two terms. For example for the two terms "chip" and "microprocessor" of the basic node n4=(chip,equipment) and the related node m=(microprocessor,equipment) we obtain the following occurrences using Google:

basic node n4:	+chip +equipment	N1=923.000
related node m:	+microprocessor +equipment	N2=203.00
corelation of the nodes n4 and m:	+chip +microprocessor +equipment	N3=71.100

The weight is computed by N3/(N1+N2–N3)=0.067. Because of very small values we normalize against the up to this point largest value as 98%.

Step 4: Insert a basic node and its related nodes into the ontology index

Let n=(t,c) the node to be inserted by a method *insert(n)*. We create a new node n, if n does not exist and the following case fails. If there exists a node k=(t,c1) with a category c1 different to the given category c, then we have to adjust the relationship between c and c1. If c (c1) is a broader term of c1 (c) we define c (c1) as the new category of node k.

In our example we try to insert the following basic nodes n1, n2, n3, n4 and their corresponding related nodes m containing related terms as computed in step 2:

n1=(chip,fecal matter)	m=(cow chip,fecal matter)	with r=m is synonym of n1
	m=(dung,fecal matter)	with r=m is hypernym of n1

…

Step 5: Create edges between a basic node and its related nodes

Let n'=insert(n) the basic node after insertion of basic node n into the ontology index. The following cases describe the creation of a new edge for each type of relationship defined in definition 1.

For all m with r=synonym or r=hypernym or r=holonym (e.g. m is synonym of n' (n), etc.) we have to create a new edge of the given type. Let m' the related node after insertion of related node m into ontology index, then E = E + {e=(n',m',is_[r]_of,1.0)}.

For all m with r=hyponym or r=meronym (e.g. m is hyponym of n' (n), etc.) we have to create a new edge of the inverse relationship r'=hypernym or r'=holonym (e.g. n'(n) is hypernym of m, etc) according to the given relationship. Let m' the related node after insertion of related node m into ontology index, then E=E+{e=(n',m',is_[r']_of,sim(n',m')}

Step 6: Adjust the weights of new edges and existing edges

Let x a basic node, y a corresponding related node and e a new edge between x and y. Let z a parent node (child node) of x and y where z – x and z – y have the same relationship type.

First, we assume x is a synonym of y and z is a hypernym (holonym, hyponym, meronym) of x and y. Then we define sim(x,z) = sim(y,z) = max{sim(x,z), sim(y,z)}.

Second, we assume x is a hypernym of y. If z is a hypernym (holonym) of x and y, then we define sim(z,y) = sim(z,x)*sim(x,y). If z is a hyponym (meronym) of x and y, then we define sim(x,z) = sim(x,y)*sim(y,z).

The cases "x is a holonym (hyponym, meronym) of y" are defined analogously.

Step 7: Connectivity

If step 5 creates no new edge between an existing node and a new node then it is necessary to create a hypernym-edge from the root node to one of the basic nodes and its do-

main node.

This algorithm based on a WordNet-style ontology as source gives two main guarantees. (1) There are no isolated nodes (strong connectivity of the graph). (2) A long path between two nodes v and w can not be more relevant than a short path between v and w.

4 Evaluation of Semantic Similarity Search Conditions using Ontology Indexes

In the XXL Search Engine we are using two ontology indexes, one over names of XML elements and XML attributes called NOI (element name ontology index) and a second over keywords extracted from the contents of XML elements called COI (content ontology index). In addition, there are an element path index (EPI) for evaluating path expressions and an element content index (ECI) for evaluating element content conditions.

To find similar terms for a given search term within an ontology index, we proceed two steps. First, we are looking for the node k containing the search term. This takes time O(log n) where n is the number of nodes. Second, we traverse the ontology index starting at the node k in a breadth-first manner. A term of a node w is similar to the given term of node k, if there is a path between w and k which either contains only edges of types hypernym and synonym or contains only edges of types hyponym and synonym. Crossing the root node is not allowed, because this would change the domain. The second step takes time O(n).

The worst-case runtime for this algorithm is O(n log n). The search for related terms using the ontology index can be restricted by the number of edges followed in one breadth-first search process or by a threshold for the similarity score of the related terms found in the ontology.

A relevance computation for a query result is described in detail in [TW02]. In this section we only present the application of the index structure to evaluate an *elementary similarity path expression* and an *elementary similarity content condition* within an XXL query.

Case I: elementary similarity path expression within an XXL query

A subquery which is interested in all documents containing an element named "chair" or a related element name can be expressed as Expr1: "~chair".

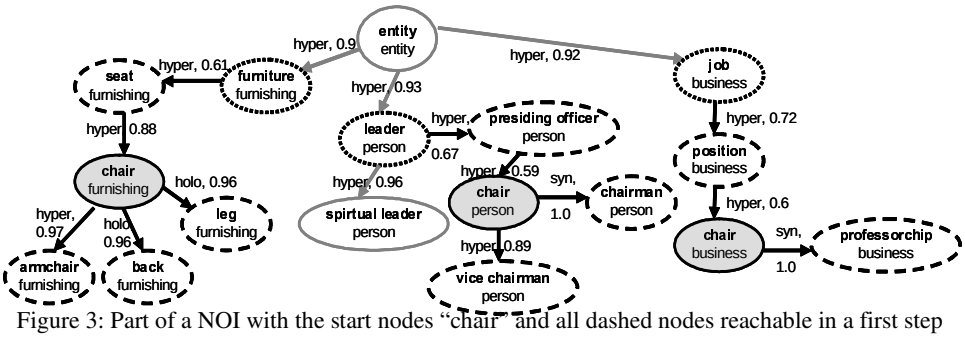


Figure 3: Part of a NOI with the start nodes "chair" and all dashed nodes reachable in a first step and all dotted nodes reachable in a second step of the breadth-first search algorithm

Consider the graph shown in Figure 3 as part of a NOI with the corresponding relationships and fictitious weights.
In a first step we compute all related terms using the NOI to the given term "chair". All similarity scores are computed as defined in Definition 4 in section 3.1. The algorithm provides the following information:

- start nodes: chair (furnishing): 1.0; chair (person): 1.0; chair (business): 1.0
- first step: armchair (furnishing): 0.97; back (furnishing): 0.96; leg (furnishing): 0.96,…
- second step: furniture (furnishing): 0.56; job (business): 0.49; leader (person): 0.48; …

In a second step we group these nodes by their domains. According to the NOI of Figure 3 we have three children of the root node (entity,entity), namely (furniture,furnishing), (leader,person) and (job,business). Therefore in our example we have information of three different domains.

- furnishing: chair 1.0; armchair 0.97; …
- person: chair 1.0; chairman 1.0; …
- business: chair 1.0; position 0.6; …

In the third step we create for each domain a new elementary query according to the given related terms and its similarities provided by the NOI.

- Expr1': "chair|armchair|back|leg|seat|furniture"
- Expr1'': "chair|chairman|presiding officer|vice chairman|leader"
- Expr1''': "chair|position|professorship|job"

In the last step we use the element path index (EPI) and the similarity scores from the first step to compute a set of result nodes and its local relevance values for each new domain-oriented elementary query Expr1', Expr1'', and Expr1'''.

Case II: elementary similarity content condition
A subquery wich is interested in all elements named "title" with content about algorithms for XML written in Java and not written in C++ can be expressed by this expression:

Expr2: "title ~ '(java or xml) and algorithm and not c++'"

In the first step we decompose the complex content condition into a binary *LogicTree* where an inner node contains an operator and a leaf node contains a search string.
In the next step for each search string (leaf node) we call the content ontology index (COI) for related terms and their categories. Then, analogously to step 1 and step 2 of case I above we group them by their domain. In the fourth step we consider all terms of one domain for domain-oriented search. For each search string we combine its related terms in a disjunctive manner. For each leaf we use the element content index (ECI) implemented in Oracle9i and its Contains-function to find appropriate elements. For the relevance computation for one found element we multiply the COI-based similarity score and the ECI-based tf*idf-based similarity score. Finally, we evaluate the LogicTree.

5 Conclusion

Currently ontologies considered as shared conceptualizations of some domain are increasingly seen as the key to further automation of information processing. Although many approaches for representing and applying ontologies have already been devised, they have not found their way into search engines for querying XML data. In this paper we have shown that ontologies using categories and differentiated semantic relationships

are useful to improve the similarity search on XML data. Our approach is implemented in the XXL search engine.

6 References

[BEH+02] E. Bozsak, et. al.: KAON - Towards a large scale Semantic Web. In: Proc. of EC-Web 2002. LNCS, Springer, 2002.

[BHP94] M.W. Bright, A.R. Hurson, S. Pakzad: Automated Resolution of Semantic Heterogeneity in Multidatabases. ACM Transactions on Database Systems, 19(2) 1994, pp. 212-253.

[BM99] T. Bench-Capon, G. Malcolm: Formalising Ontologies and Their Relations. Proc. of the 10th DEXA Conf. 1999, pp: 250-259.

[BR99] R. Baeza-Yates, B. Ribeiro-Neto: Modern Information Retrieval, Addison Wesley, 1999.

[DEFS99] S. Decker, M. Erdmann, D. Fensel, R. Studer: OntoBroker: Ontology based Access to Distributed and Semi-Structured Information. Semantic Issues in Multimedia Systems, Proc. of DS-8, Kluwer, 1999, pp. 351-369.

[FAF+02] R. Feldman, Y. Aumann, M. Finkelstein-Landau, E. Hurvitz, Y. Regev, A. Yaroshevich: A Comparative Study of Information Extraction Strategies. Proc. of the CICLing Conf. 2002.

[FDH98] R. Feldman, I. Dagan, H. Hirsh: Mining Text Using Keyword Distributions. Journal of Intelligent Information Systems 10 (1998), pp. 281-300.

[Gru93] T.R. Gruber: Towards Principles for the Design of Ontologies used for Knowledge Sharing. Proc. of the International Workshop on Formal Ontology, 1993.

[Gua98] N. Guarino: Formal Ontology in Information Systems. Proc. of FOIS'98, IOS Press.

[Hor02] I. Horrocks: DAML+OIL: A Reasonable Web Ontology Language. Proc. of the 8th EDBT Conf. 2002, pp. 2-13.

[KI90] H. Kimoto, T. Iwadera: Construction of a Dynamic Thesaurus and its Use for Associated Information Retrieval. Proc. of the SIGIR Conf. 1990, pp. 227-240.

[KKC94] O. Kwon, M.-C. Kim, K.-S. Choi: Query Expansion Using Domain Adapted, Weighted Thesaurus in an Extended Boolean Model. Proc. of the CIKM Conf. 1994, pp. 140-146.

[KKS01] L. Kerschberg, W. Kim, A. Scime: A Semantic Taxonomy-Based Personalizable Meta-Search Agent. Proc. of the WISE Conf. 2001.

[MWK00] P. Mitra, G. Wiederhold, M.L. Kersten: Articulation of Ontology Interdependencies Using a Graph-Oriented Approach, Proc. of the 7th EDBT Conf., Constance, Germany, 2000.

[PM98] M.P. Papazoglou, S. Milliner: Subject-based Organization of the Information Space in Multi-Database Networks. Proc. of the 10th CAiSE Conf. 1998.

[RN95] S. Russel, P. Norvig: Artificial Intelligence - A Modern Approach, Prentice Hall, 1995

[SAD+00] S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Mädche, H.-P. Schnurr, R. Studer: Semantic Community Web Portals, 9th WWW Conference, 2000.

[SemWeb] World Wide Web Consortium: Semantic Web Activity, http://www.w3.org/2001/sw/

[SM00] S. Staab, A. Maedche: Ontology Engineering beyond the Modeling of Concepts and Relations. Proc. of the 14th ECAI Conf., Workshop on Applications of Ontologies and Problem-Solving Methods, 2000.

[STW01] S. Sizov, A. Theobald, G. Weikum: Ähnlichkeitssuche auf XML-Daten. In: A. Heuer, F. Leymann, D. Priebe (Eds.): Datenbanksystem in Büro, Technik und Wissenschaft (BTW) 2001, 9. GI-Fachtagung, Oldenburg, In: Informatik aktuell, pp 364-383, Springer Verlag, 2001.

[TW00] A. Theobald, G. Weikum: Adding Relevance to XML. In: D. Suciu, G. Vossen (Eds.): The World Wide Web and Databases. Lecture Notes in Computer Science 1997, pp 105-124, Berlin: Springer, 2001.

[TW02] A. Theobald, G. Weikum: The Index-based XXL Search Engine for Querying XML Data with Relevance Ranking. In: Ch. S. Jensen, K. G. Jeffery, J. Pokorny, S. Saltenis, E. Bertino, K. Böhm, M. Jarke (Eds.): EDBT Conf. 2002. LNCS 2287, Berlin: Springer, 2002.

[WN98] C. Fellbaum (ed.): WordNet: An Electronic Lexical Database. MIT Press 1998.

Type Checking in XOBE

Martin Kempa, Volker Linnemann
Universität zu Lübeck
Institut für Informationssysteme
Osterweide 8
D-23562 Lübeck, Germany
email: {kempa|linnemann}@ifis.uni-luebeck.de

Abstract: XML is the upcoming standard for internet data. Java is the most important programming language for internet applications. Nevertheless, in today's languages and tools there is no smooth integration of Java and XML. The **X**ML **OB**j**E**cts project (XOBE) at the University of Lübeck addresses this mismatch by defining XML objects by XML schemas and by making them to first-class data values. In XOBE, the distinction between XML documents and XML objects no longer exists. Instead, a running XOBE program works only with XML objects. XML documents in text form with explicit tags exist only for communicating with the outside world. This approach allows to check the validity of all XML objects within a program statically at compile time. This is accomplished by XML constructors. Previously generated XML objects are inserted in these constructors such that the validity can be checked at compile time. This paper concentrates on the type checking algorithm in XOBE which is used, among others, for checking the correctness of assignment statements involving XML objects. The type checking algorithm assures that all XML objects that can occur dynamically on the right hand side of an assignment statement are objects that can be assigned to the variable on the left hand side. This type checking is done statically without running the program. The algorithm is based upon regular hedge grammars and regular hedge expressions.

1 Introduction

In the last years the Extensible Markup Language (XML) [W3C98b] has become the standard data format of the Internet. Many different XML-based markup languages have been developed, the usages of which range from publishing documents on web sites to the exchange of data. Committees like the World Wide Web Consortium (W3C) enforce the process of developing XML-related standards, like XPointer, XPath, XSLT, XQuery and software vendors introduce XML-based tools or extend their current products to be accessible via XML. Recently the concept of web services realizing software components over the Internet instead of stand alone web applications became popular.

Today, the implementation of most web applications and web services is realized by standard programming languages like Java [AG98] or Visual Basic. Modern web applications and web services generate XML structures intensively. The content of these dynamic

structures is assembled at run time, in contrast to static web pages, which do not change at run time. For the creation of dynamically generated structures technologies like CGI [Gai95], Java Servlets [Wil99], Java Server Pages [PLC99, FK00] or JAXB [Sun01] are used.

Using these technologies guarantees the correctness of dynamically generated structures only to a very limited extend. XML structures should not only be well-formed, but should also be valid according to an underlying DTD [ABS00] or XML schema [W3C01b], which we call schema in the following, defining the used markup language. Instead, the validity must be "proven" dynamically by appropriate test runs. Moreover, some techniques differentiate between XML strings and XML objects requiring to switch between these two notions by methods called marshalling and unmarshalling. This leads to a serious mismatch between objects in an object oriented programming language and XML structures.

The **X**ML **OBjE**cts project (XOBE) [LK02] overcomes the differences between XML structures as strings and corresponding objects by an extension of the object oriented programming language Java. In other words, when using XML syntax in XOBE, it always denotes XML objects, i.e. generating and analyzing XML is done conceptually only on the basis of objects. Therefore XOBE introduces a class for every element type of a used schema, but does not generate these classes explicitly as Java classes. Instead of that they can be used like built-in data types. They are defined in such a way that the generation of XML objects is done in a syntax oriented manner allowing to check most portions of the property validity, which we call static validity, of all generated XML structures, i.e. XML objects, at compile time.

The present paper focuses on the XOBE type system. The main problem of the XOBE type system is in deciding if an XML object is allowed at the position it appears. This decision problem for subtyping is algorithmically difficult. We introduce an algorithm which checks the subtype relationship of XML objects. The algorithm can be viewed as an improvement of Antimirov's algorithms for the decision problem of regular expressions. Even on XOBE applications that involve quite large types, such as the complete schema of XHTML, our algorithm completes in reasonable time.

Our algorithm enables XOBE to guarantee static validity at compile time which implies the following advantages:

1. XOBE programs are more efficient because we avoid expensive run time checks to guarantee validity.
2. XOBE programs are more reliable because we can omit the programming of recovery procedures which are needed if run time checks fail.
3. XOBE enables a faster development of implementations, because we can avoid extensive test runs, which are necessary to make the validity of the generated XML documents feasible.
4. XOBE improves the maintenance of web services and web applications because it leads to a simpler source code structure.

The paper is organized as follows. In the next section, we give an introduction to the programming with XML objects. In Section 3, we describe the connection between XML objects and regular hedge expressions and introduce subtyping. In Section 4, we present our subtyping algorithm, the main part of the paper. Section 5 explains our implementation techniques and Section 6 discusses some performance measurements. We survey related work in Section 7, i.e. we summarize the state of the art of programming web applications and web services. Section 8 concludes the paper and gives an outlook on future work.

2 XML Objects

In this section we introduce the syntax and semantics of XML Objects (XOBE) briefly. A more detailed introduction can be found in [KL02]. XOBE extends the object-oriented programming language Java by a compile time validation mechanism for dynamically generated XML structures.

Constructing XML Objects

XML structures, which are trees corresponding to a given schema, are represented by *XML objects* in XOBE. Therefore XML objects are first-class data values that may be passed and stored like any other data values. The given schema is used to type different XML objects.

XOBE programs import schema definitions, declared by an import statement `ximport`, to use *XML objects* in the source code. The schema SIF is imported with "`ximport SIF.xsd;`" for example. The classes of these objects are not generated explicitly as Java source code. In fact the element declarations and type definitions in the imported schema define the available classes directly. Every element declaration and every type definition in the schema corresponds to an implicit XML object class. In other words, this means that XML objects are instances of an element declaration or a type definition of an imported schema. Therefore XML objects are XML structures having the corresponding element or elements as the root node.

New XML objects can be created in XOBE programs by expressions which we call *XML constructors*. These expressions are denoted in valid XML syntax where other XML objects can be inserted in places which are allowed according to the imported schema. These values are separated from the surrounding XML syntax by braces.

With the concept of XML objects there is no distinction between the string representation and the object representation of XML structures in XOBE. Thus XML structures in a program always denote objects. Using only XML objects guarantees the property well-formedness as well as static validity for the dynamically generated XML objects at compile time.

We will explain our concepts by the following running example. A web service implementing a shopping application communicates with the outside world using an XML data format. The data format called Shop Interchange Format (SIF) is given in Appendix A.

Our examples focus on the class `Cart` depicted in Figure 1. The class has a member field

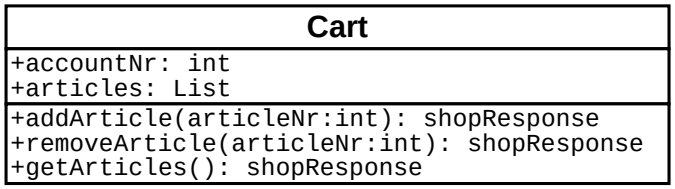


Figure 1: Class Cart

`accountNr` of type `int` which identifies the customer to which this cart belongs and a member field `articles` of class `List` saving already selected items. Further, the class has the three methods `addArticle`, `removeArticle`, and `getArticles`. They make it possible to add and remove certain articles and to display already selected articles.

The following XOBE method generates an SIF `shopResponse`-object containing the account number being taken from the member field `accountNr`.

```
shopResponse removeArticle(int articleNr){
   request done;
   shopResponse response;
   if (this.articles.remove(articleNr))
      done = <request>processed</request>;
   else
      done = <request>fail</request>;
   response = <shopResponse><shoppingCart>
                 <account>{this.accountNr}</account>
                  {done}
              </shoppingCart></shopResponse>;
   return response;
} // removeArticle
```

Listing 1: Method `removeArticle`

We allow `int`-values in places where `String`-values are expected according to the underlying schema. The `int`-value is converted automatically to its decimal notation as a `String`-value by calling method `toString`. It should be obvious that method `removeArticle` is guaranteed statically to generate valid SIF.

A XOBE program generates XML only by XML constructors, i.e. there is no string representation during generation. String generation is necessary only when the document is communicated to the outside world, for example as the result of a Java servlet. Only for this purpose a method `toString` is provided for XML objects.

Although XOBE can ensure most predicates of the property validity statically, runtime checks are necessary in some exceptions. Similar to an array of constant length, where the index has to be in the declared range, the number of occurrences of a specific element has to be between the values set by the attributes `minOccurs` and `maxOccurs` in the defining schema. Additional runtime checks are necessary for identity constraints, restricted string types and facets on numeric types.

Accessing XML Objects

XOBE incorporates XPath [W3C99] for accessing XML objects in XOBE programs. XPath provides a mechanism to express path expressions extracting some nodes in an XML structure. A path expression in general consists of a *location path*, selecting subnodes of a given *context node*. An additional *node test* restricts the selected nodes to specified element names. Further restrictions can be performed with optional *predicates*. In XOBE, the context node is denoted by an XML object variable. XPath expressions in XOBE return a list of nodes, which is regarded as an XML object.

Listing 2 shows the implementation of method `processRequest` processing an incoming shop request. The request is passed to the addressed cart, which returns the resulting response. Note that a global variable `allCarts` is used which gives access to all registered `Cart`-objects.

```
1  shopResponse processRequest(shopRequest rq) {
2     Cart c;
3     shopRequest.shoppingCart sc;
4     sc = rq/shopRequest/shoppingCart[1];
5     c = allCarts.get(sc/account[1]);
6     if (sc/add.getLength() == 1) return c.addArticle(sc/add[1]);
7     else if (sc/remove.getLength() == 1) return c.removeArticle(sc/remove[1]);
8     else if (sc/get.getLength() == 1) return c.getArticles();
9  } // processRequest
```

Listing 2: Method `processRequest`

From the incoming request `rq` the method extracts the shopping cart and assigns it to variable `sc`. Afterwards the targeted cart is chosen from the set of available carts `allCarts`. A nested conditional statement differentiates the three possible requests and calls the suitable methods. The required parameters are gathered from the request.

In the method `processRequest` we use XPath constructs for accessing XML object content. Using the path notations (4-8) it is possible to navigate through XML objects. Because the XPath node test returns a list of resulting XML objects, we have to access the first item of the sequence with the XPath integer predicate (4-7). We determine the size of such a list using method `getLength` (6-8).

The impact of our approach on the host language Java is an XML consistent extension of the type system. Because of the outstanding role of XML in the web application and web services programming world, and maybe the whole software development world in the future, this seems to be a consequent step. We believe that the trade-offs between the extension of an existing programming language on one hand and the approach of defining a new programming language around XML on the other are minimal. Instead the benefits of using already developed code are significant.

3 Basic Definitions

As seen in the last section XOBE allows XML syntax in expressions, assignments and method parameters. During compilation the XOBE system verifies the correctness of the

assignment in two steps. First it determines the types of the right and left hand sides using type inference. Secondly, the subtype relationship of the inferred types is checked by a subtyping algorithm.

In XOBE we formalize and represent types as regular hedge expressions representing regular hedge languages [BKMW01]. Consequently a schema is formalized and represented internally by a regular hedge grammar.

Definition 1 (regular hedge grammar)
A regular hedge grammar is defined by $G = (T, N, s, P)$ with a set $T = B \cup E$ of terminal symbols, consisting of simple type names B and a set E of element names (Tags), a set N of nonterminal symbols (names of groups and complex types), a start expression s and a set P of rules or productions of the form $n \rightarrow r$ with $n \in N$ and r is a regular hedge expression over $T \cup N$.

The rules in the production set P have to fulfill the following two constraints: [1]

1. Recursive nonterminals may appear in tail positions only.
2. Recursive nonterminals must be preceded by at least one non-nullable expression.

□

A non-nullable expression is a regular hedge expression which does not contain the empty hedge.

We define the regular hedge expressions, referred to in short as regular expressions, similar to the notation used in [W3C01a].

Definition 2 (regular hedge expression)
Given a set of terminal symbols $T = B \cup E$ and a set N of nonterminal symbols, the set Reg of regular hedge expressions is defined recursively as follows:

$$
\begin{array}{rll}
\emptyset & \in Reg & \text{the empty set,} \\
\epsilon & \in Reg & \text{the empty hedge,} \\
b & \in Reg & \text{the simple types,} \\
n & \in Reg & \text{the complex types,} \\
e[r] & \in Reg & \text{the elements,} \\
r|s & \in Reg & \text{the regular union operation,} \\
r, s & \in Reg & \text{the concatenation operation, and} \\
r* & \in Reg & \text{the Kleene star operation.}
\end{array}
$$

for all $b \in B, n \in N, e \in E, r, s \in Reg$. □

As an example we formalize the schema SIF (Appendix A) introduced in the last section as a regular hedge grammar as $G = (B \cup E, N, (shopRequest|shopResponse), P)$ with:

$$B = \{\text{integer}, \text{string}, \text{t\_request}\},$$
$$E = \{\text{shopRequest}, \text{shopResponse}, \text{shoppingCart}, \text{account}, \text{add},$$

[1] The two constraints ensure regularity.

$$
\begin{aligned}
&\text{remove}, \text{get}, \text{request}, \text{items}, \text{article}, \text{description}\},\\
N = \{&t\_shopRequest, t\_cartRequest, t\_shopResponse, t\_cartResponse,\\
&t\_items, shopRequest, shopResponse, t\_shopRequest.shoppingCart,\\
&t\_shopResponse.shoppingCart, account, add, remove, get, request,\\
&items, article, description\}, \text{ and}\\
P = \{ \quad & t\_shopRequest \rightarrow t\_shopRequest.shoppingCart;\\
& t\_cartRequest \rightarrow (account, (add|remove|get));\\
& t\_shopResponse \rightarrow t\_shopResponse.shoppingCart;\\
& t\_cartResponse \rightarrow (account, request, (items|\epsilon));\\
& t\_items \rightarrow (article*, (description|\epsilon));\\
& shopRequest \rightarrow \text{shopRequest}[t\_shopRequest];\\
& t\_shopRequest.shoppingCart \rightarrow \text{shoppingCart}[t\_cartRequest];\\
& account \rightarrow \text{account}[\text{integer}];\\
& add \rightarrow \text{add}[\text{integer}];\\
& remove \rightarrow \text{remove}[\text{integer}];\\
& get \rightarrow \text{get}[\epsilon];\\
& shopResponse \rightarrow \text{shopResponse}[t\_shopResponse];\\
& t\_shopResponse.shoppingCart \rightarrow \text{shoppingCart}[t\_cartResponse];\\
& request \rightarrow \text{request}[\text{t\_request}];\\
& items \rightarrow \text{items}[t\_items];\\
& article \rightarrow \text{article}[\text{integer}];\\
& description \rightarrow \text{description}[\text{string}]\}.
\end{aligned}
$$
[23]

As in XML Schema we do not demand that the set of element names E and the set of complex types N have to be disjunct. In this paper we use different fonts to separate element names from complex type names. Mixed content is formalized as a simple type string. We deal with attributes similar to elements, but with a specially marked name, rewriting them with regular hedge constructors. Any-types are rewritten to a regular union of all declared element types as well.

As mentioned above XOBE infers at compile time both types of the right and left hand sides of an assignment. Because all variables have to be declared, the type inference of variables is simple. In our example of Listing 1 a variable `done` is declared of type *request* and a variable `response` of type *shopResponse*. Based on the variable types, the type of the whole XML constructor on the right hand side can be inferred. In our example it is shopResponse[shoppingCart[account[integer], *request*]].

After inferring the types of the left and right hand sides, the XOBE type system checks if the type of the right hand side is a subtype of the type of the left hand side. For this

[2] Because the comma (,) is used as concatenation operation in regular expression, we use the semicolon (;) as separator in sets where regular expressions appear as elements.

[3] Element name `shoppingCart` has two different types which we have to distinguish.

example XOBE has to check the so-called *regular inequality*

$$\text{shopResponse}[\text{shoppingCart}[\text{account}[\text{integer}], request]] \leq shopResponse$$

where $\leq$ stands for the subtype relationship. Note, that the name $shopResponse$ on the right hand side stands for the complex type $shopResponse$. The name shopResponse on the left hand side is an element name.

4 Subtyping Algorithm

Checking the subtype relationship between two regular hedge expressions is the main task in proving type correctness of XOBE programs. For this we adopt the Antimirov algorithm [Ant94] for checking inequalities of regular expressions and extend it to the hedge grammar case. The idea behind Antimirov's algorithm is that for every invalid regular inequality there exists at least one reduced inequality which is *trivially inconsistent*. An inequality is trivially inconsistent if the empty hedge is in the language represented by the regular expression on the left hand side but not in the language represented by the right hand side regular expression.

The algorithm operates as follows: It takes the regular inequality to prove as argument and retrieves the leading simple type names and element names from the left hand side regular expression using operation `leadingNames`. The operation leading names is defined as follows.

Definition3 (leading names)
Given a regular expression $r \in Reg$ the operation $leadingNames$ returns the set of all leading terminal symbols:

$$\begin{aligned}
leadingNames(\emptyset) &= \{\} \\
leadingNames(\epsilon) &= \{\} \\
leadingNames(e[r]) &= \{e\} \\
leadingNames(b) &= \{b\} \\
leadingNames(n) &= leadingNames(r) \text{ with } n \rightarrow r \in P \\
leadingNames(r|s) &= leadingNames(r) \cup leadingNames(s) \\
leadingNames(r,s) &= \begin{cases} leadingNames(r) \cup leadingNames(s) & \text{if } isNullable(r) \\ leadingNames(r) & \text{if } \neg isNullable(r) \end{cases} \\
leadingNames(r*) &= leadingNames(r)
\end{aligned}$$

with $b \in B, n \in N, e \in E, r, s \in Reg$. □

It is defined recursively and returns the leading simple type names and element names. If a complex type name occurs the operation uses the production definition. For a regular concatenation the operation needs the $isNullable$ predicate to check the empty hedge inclusion.

For each determined name the algorithm tries to reduce both sides of the inequality by this name. The resulting reduced inequalities are simpler than the starting inequality in the majority of cases and can be checked by a recursive application of the algorithm. The algorithm tracks already treated inequalities in a set of inequalities, which is empty in the beginning. This ensures termination if we encounter the same inequality later on. We do not prove subtyping directly, instead we calculate the set of all possible reduced inequalities of the given inequality. If we receive a trivially inconsistent inequality we conclude that the given inequality is incorrect. In all other cases we assume that the given inequality holds. This is a standard proceeding in subtyping algorithms of recursive types.

There are two different results of our recursive algorithm. First the algorithm responds false if the inequality in question is trivially inconsistent using the `isNullable` predicate. Secondly, the algorithm terminates with true when it processes an inequality which is in the set of already processed inequalities. This means that our algorithm cannot produce any new inequality in this branch of recursion. If there are still inequalities to derive, the algorithm continues with the operation `partialDerivatives` which is explained after the algorithm definition.

Definition 4 (subtyping algorithm)
Given a set A of already processed inequalities the algorithm to prove $r \leq s$ is defined by the following pseudo code:

```
bool prove(r ≤ s, A) {
  if ((isNullable(r) && ~isNullable(s)) || s == ∅) return false;
  elsif ((r ≤ s) ∈ A) return true;
  else {
    ok := true; pd := ∅; ns := leadingNames(r);
    forall (n ∈ ns) pd := pd ∪ partialDerivatives(n,r ≤ s);
    A := A ∪ {r ≤ s};
    forall ((r1 ≤ s1) ∨ (r2 ≤ s2) ∈ pd)
      ok := ok && (prove(r1 ≤ s1,A) || prove(r2 ≤ s2,A));
    return ok;
  } // else
} // prove
```

Listing 3: Subtyping Algorithm

□

Antimirov introduces so-called *partial derivatives of regular expressions* to express reduced regular expressions. A partial derivative reduces a regular expression by a given type name or element name. In the hedge grammar setting we modify partial derivatives concerning type names and element names. For example, if we have the regular expression $(\text{account}[\text{integer}], \text{request}[\text{t\_request}])$ and calculate its partial derivatives with respect to the given element name account, we receive the result $\{(\text{integer}; \text{request}[\text{t\_request}])\}$. The result is a set in general, because we can get multiple derivatives for a given regular expression. The elements of the set are pairs corresponding to the two dimensions in a regular hedge, the parent-child dimension and the sibling dimension. In the example the first component of the pair is the type of the content of element account. The second is the regular expression reduced by element account.

Additionally Antimirov introduces so-called *partial derivatives of regular inequalities* to express reduced inequalities. In the hedge grammar case these become more complicated. Because the partial derivatives of regular expressions are pairs we have to perform a set-

theoretic theorem observed by Hosoya, Vouillon and Pierce [HVP00].

Theorem 5 (subset relation on Cartesian product)
Given some sets $a, b, c_1, \ldots, c_n, d_1, \ldots, d_n$ the following holds:

$$a \times b \subseteq (c_1 \times d_1) \cup \cdots \cup (c_n \times d_n)$$
$$\Leftrightarrow$$
$$(a \subseteq \cup^{i \in I_1} c_i \vee b \subseteq \cup^{i \in \overline{I}_1} d_i) \wedge \cdots \wedge (a \subseteq \cup^{i \in I_{2^n}} c_i \vee b \subseteq \cup^{i \in \overline{I}_{2^n}} d_i)$$
$$\text{with } \mathcal{P}(\{1,\ldots,n\}) = \{I_1, \ldots, I_{2^n}\} \text{ and } \overline{I}_i = \{1,\ldots,n\} \backslash I_i \,.$$

As we can see, this theorem reduces the subset relation on Cartesian products to a subset relation on sets.

Because types can be interpreted as sets of values, we can write the following relation emerging after reducing the left and right hand sides of an inequality by the leading name n

$$(c_r \times r_r) \subseteq (c_s^1 \times r_s^1) \cup \cdots \cup (c_s^k \times r_s^k) \text{ with}$$
$$partialDerivatives(n,s) = \{(c_s^1; r_s^1), \ldots, (c_s^k; r_s^k)\}$$

for every $(c_r; r_r) \in partialDerivatives(n,r)$. To this relation we can apply the given theorem and receive

$$(c_r \subseteq \cup^{i \in I_1} c_s^i \vee r_r \subseteq \cup^{i \in \overline{I}_1} r_s^i) \wedge \cdots \wedge (c_r \subseteq \cup^{i \in I_{2^k}} c_s^i \vee r_r \subseteq \cup^{i \in \overline{I}_{2^k}} r_s^i)$$
$$\text{with } \mathcal{P}(\{1,\ldots,k\}) = \{I_1, \ldots, I_{2^k}\} \text{ and } \overline{I}_i = \{1,\ldots,k\} \backslash I_i \text{ and}$$
$$partialDerivatives(n,s) = \{(c_s^1; r_s^1), \ldots, (c_s^k; r_s^k)\}.$$

In the definition of the partial derivatives of regular inequalities we collect the disjunctions separately in a set. Additionally we rewrite the relation using the regular expression operators regular union $|$ instead of union on sets $\cup$ and inequality $\leq$ instead of subset on sets $\subseteq$.

Definition 6 (partial derivatives of regular inequality)
Given a regular inequality $r \leq s$ with $r, s \in Reg$ and a terminal symbol $n \in T$ the partial derivatives of that inequality is defined as:

$$partialDerivatives(n, r \leq s) = \{(c_r \leq \bigg|^{i \in I} c_s^i) \vee (r_r \leq \bigg|^{i \in \overline{I}} r_s^i) |$$
$$(c_r; r_r) \in partialDerivatives(n,r) \wedge$$
$$partialDerivatives(n,s) = \{(c_s^1; r_s^1), \ldots, (c_s^k; r_s^k)\},$$
$$I \in \mathcal{P}(\{1,\ldots,k\}) \text{ and } \overline{I} = \{1,\ldots,k\} \backslash I\}$$

□

A partial derivative of a regular inequality is a set the elements of which have the form of two inequalities connected with the boolean operation or. This means for our algorithm, that we can apply our procedure `prove` recursively.

In the remaining section we apply our subtyping algorithm to a small example. Consider the example where we have the two types $r \equiv (description, (\text{account}[\text{integer}], description)*)$ and $s \equiv ((\text{description}[\text{string}], \text{account}[\text{integer}])*, description)$ which we denote with r and s for a concise description. We want to check the regular inequality $r \leq s$ in the following, for which we start the subtyping algorithm with an empty set of inequalities $A = \{\}$.

In the first recursion the algorithm calculates the set of leading names $ns = \{\text{description}\}$. Additionally the set A of already processed inequalities is enlarged to $A := A \cup \{r \leq s\}$. We get

$$partialDerivatives(description, r) = \{(\text{string}; (\text{account}[\text{integer}], description)*)\}$$
$$partialDerivatives(description, s) = \{(\text{string}; \text{account}[\text{integer}], s); (\text{string}; \epsilon)\}$$

and Definition 6 leads to

$$\begin{aligned} pd = \{&(\text{string} \leq \text{string} \mid \text{string} \vee && (1)\\ &\quad (\text{account}[\text{integer}], description)* \leq \emptyset); && (2)\\ &(\text{string} \leq \text{string} \vee && (3)\\ &\quad (\text{account}[\text{integer}], description)* \leq \epsilon); && (4)\\ &(\text{string} \leq \text{string} \vee && (5)\\ &\quad (\text{account}[\text{integer}], description)* \leq \text{account}[\text{integer}], s); && (6)\\ &(\text{string} \leq \emptyset \vee && (7)\\ &\quad (\text{account}[\text{integer}], description)* \leq (\text{account}[\text{integer}], s) \mid \epsilon)\}. && (8) \end{aligned}$$

The inequalities 1, 3 and 5 evaluate trivially to true, while inequalities 2, 4, 6 do not hold. Because inequality 7 is false, inequality 8 has to be checked in another recursion of our algorithm.

Let be $r' \equiv (\text{account}[\text{integer}], description)*$ and $s' \equiv (\text{account}[\text{integer}], s) \mid \epsilon)$ for further description. For proving inequality 8 the algorithm calculates the sets of leading names $ns = \{\text{account}\}$. Again the set of inequalities is enlarged to $A := A \cup \{r' \leq s'\}$. The following partial derivatives are generated during the algorithm:

$$\begin{aligned} pd = \{&(\text{integer} \leq \text{integer} \vee && (9)\\ &\quad r \leq \emptyset); && (10)\\ &(\text{integer} \leq \emptyset \vee && (11)\\ &\quad r \leq s)\}. && (12) \end{aligned}$$

It is easy to see, that inequality 9 holds and inequalities 10 and 11 evaluate to false. The last inequality 12 is true, because it is already in our set of already processed inequalities

$r \leq s \in A$. Finally the algorithm accepts $r \leq s$ as correct, because we did not derive inconsistent inequalities in both branches of any disjunction.

To prove the correctness of the algorithm we use the set of all derivable inequalities, which in fact is a family of sets, one set for every recursion branch. It can be shown, that an inequality $r \leq s$ is not valid, if and only if all sets derivable from $r \leq s$ contain false. Secondly it follows that after a finitenumber of steps each set of inequalities either contains false or saturates. This property ensures termination. Compared to standard subtyping based on regular tree automata which involves the computation of automata intersection and automata complement, our algorithm is more efficient. Although our algorithm has a potential exponential inefficiency as the automata procedure, there are cases where our algorithm is exponentially faster.

5 Implementation Issues

The previous sections presented the representation of types in XOBE. We now describe our XOBE implementation architecture [Kra02], which is shown in Figure 2. Although it

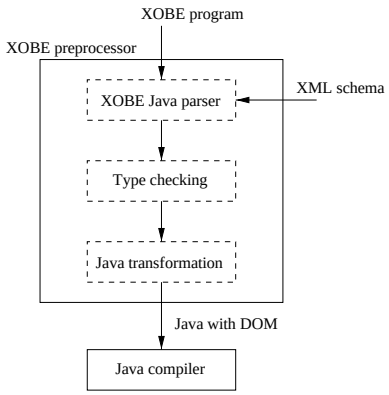


Figure 2: XOBE Java Architecture

is possible to integrate the functionality of XOBE into the Java compiler, we have chosen to implement XOBE as a Java preprocessor. The XOBE preprocessor consists of the following three components:

1. The XOBE parser,
2. the type checking analysis and
3. the transformation to standard Java code.

The XOBE parser reads the XOBE program and converts the XML portions of the program to an internal representation. The parser includes in addition to a standard Java parser

a schema parser and a slightly modified XML parser. The schema parser is necessary to scan the schemas imported by the XOBE program. The XML parser is needed to recognize the XML constructors distributed over the program source code. Because the XML constructors can include XML variables we have to modify the standard XML parser. In our implementation we utilize the Java compiler compiler JavaCC [Web02] to generate the XOBE parser. Additionally we use the XML parser Xerces [The01] to recognize the used schemas. The internal representation of the processed XOBE program is done with the Java tree builder JTB [TWP00].

In the type analysis phase the preprocessor determines whether the parsed program is well-typed or not. Well-typed in XOBE means that the processed XML objects are valid according to the declared schemas. At first the type analysis phase validates the imported schemas. Afterwards, the type check of Java expressions using XML objects, like assignments or method calls, is performed according to the description of the previous section. The type inference of XML constructors and XML variables is followed by subtyping proofs to verify the expressions. Because the type system of standard XML is strict and can be formalized by restricted regular hedge expressions, we can use the regular hedge grammar based algorithm to decide the equivalence of regular expressions for that purpose. XML Schema weakens the strictness by introducing type extension and type restriction, which requires a more sophisticated type inference strategy. The detailed description of this extended algorithm will be introduced in thesis [Kem03].

The last task the preprocessor performs is the transformation of the XOBE program into Java source code, which is accepted by the standard Java compiler. For this resulting Java code several implementation alternatives exist, depending on the XML representation. We chose the standard representation of the Document Object Model, or DOM [W3C98a], recommended by the W3Consortium. The transformation replaces the XML constructors and XPath expressions of the XOBE program with suitable DOM code. The exact transformation rules will be presented in [Kem03]. Please note that even though DOM is an untyped XML implementation not guaranteeing static validity, the transformed XML objects in the XOBE program are valid. This holds because our type checking algorithm guarantees this property. In our implementation the transformation is performed on the internal JTB representation, where we replace the subtrees representing XOBE constructs by newly created subtrees, which represent the suitable DOM code.

6 Experimental Results

In this section we present some preliminary performance measurements of the XOBE implementation. Our interests concerning the performance is twofold. First, we want to know the time which is spent for precompiling XOBE programs and especially for the type checking algorithm. Second, we measure the evaluation time of our resulting DOM-based servlets which we compare to a standard non-DOM servlet implementation. The programs are executed on a Sun Blade 1000 with two Ultra Sparc 3 (600 Mhz) processors running Solaris 8 (SunOS 5.8).

Estate is a small program which generates XHTML web pages for an estate broker. Real estates are stored in an XML file following a small non-standard schema. These files are taken as input and converted into free-standing XHTML documents.

Login is a servlet which realizes the login for an academic exercise administration system. It requests login and password and passes the input to the system.

MobileArchive is a servlet-based web application realizing a WML-connection to a medical media archive. A navigation through the archive structure similar to a file system tree is possible, in addition to a media search. Some media objects of specific formats can be viewed as well.

The first application is written as several simple iterative methods, performing a straightforward traversal of the input tree. The second application communicates over JDBC with an Informix database management system. The client input is passed to the database the respond of which is wrapped into XHTML and sent to the client. The last application is a WML connection to a media archive. The media archive is accessed through a given API. The application memorizes the position of the actual client in the structure of the archive.

We use XHTML (more precise XHTML-transitional) as schema in programs Estate and Login and WML in MobileArchive. The last two applications have been migrated from a standard Java servlet implementation. Many silly mistakes in the non-XOBE implementation have been found doing this task. Despite careful test runs of the previous implementation, many mistakes have been overlooked.

	lines of code		compile time (s)		execution time (s)	
Application	XOBE	schema	total	subtyping	standard	XOBE
Estate	158	1231	2.16	0.05	-	0.9
Login	195	1196	4.61	0.07	0.01	0.01
MobileArchive	1045	355	4.49	0.16	0.03	0.04

In the table the number of lines of the whole XOBE programs and the number of lines of the imported schemas is presented in the first two columns. In the second group of columns the table shows the time spent for precompiling the XOBE program. It includes the parsing, the type inference, the subtyping algorithm and the code transformation into standard Java, as described in Section 5. The time spent during the subtyping algorithm is shown in column "subtyping". The third column group of the table gives an impression of how the performance of the servlets is affected by the DOM-based implementation. The column "standard" shows the time which has elapsed evaluating the standard non-XOBE servlet implementation. The last column gives the running time of the XOBE program.

As indicated by the table our type checking algorithm runs at acceptable speed for these applications. Even the applications which use quite large types from the XHTML or WML schema are compiled in encouraging time. The execution speed of our DOM-based servlet implementations is slower than the standard servlets as expected but still in a convenient range.

7 Related Work

A number of approaches have been presented facilitating XML processing and generation in existing programming languages. They mainly differ in managing the structure of XML documents. Only a few similar ideas compared to our approach have been proposed.

String processing

The most elementary way to deal with XML documents is to use the string operations which are provided by the programming language, i.e. XML documents are treated as ordinary strings without any structure. The most prominent representative of this technique is given by Java Servlets [Wil99]. In former CGI scripts [Gai95] the programming language Perl [WS92] was used. The technique is rather tedious, when constant XML fragments are being generated. At compile time string operations neither guarantee well-formedness nor static validity.

Java Server Pages [PLC99] are translated by a preprocessor into Java servlets. They allow to switch between XML parts and Java parts for generating XML documents. This switching is done by special markings. Compared to string operations, this technique provides some progress especially when constant XML fragments are being generated. Java Server Pages share with string operations the disadvantage that not even well-formedness is checked at compile time.

Low-level binding

An improvement is to provide classes for nodes of an XML document tree thus allowing to access and manipulate arbitrary XML documents by object-oriented programming. Representatives of this approach, sometimes called low-level binding, are the Document Object Model (DOM) [W3C98a] and JDOM [JDO]. They are widely accepted and supported. It is the only standardized and language independent way for XML processing. Constant XML fragments can be programmed in a pure object-oriented manner, which is rather tedious, or by parsing an XML fragment into the object structure, which requires runtime validation. Low-level bindings ensure well-formedness of dynamically generated documents at compile time, but defer validation until runtime.

JavaScript [Net97] is embedded in HTML and runs on the browser side. It allows, among others, to generate HTML parts dynamically. This can be done either on a pure string basis or on the basis of the DOM.

High-level binding

Recently a series of proposals [Bou02], called high-level bindings, have been presented. With Sun's JAXB, Microsoft's .Net Framework, Exolab's Castor, Delphi's Data Binding Wizard, Oracle's XML Class Generator [Sun01, Mic01, Exo01, Bor01, Ora01] we only mention the better known products. These approaches deal with the assumption that all processed documents follow a given schema. This description is used to map the document structure onto language types or classes, which reproduce directly the semantics intended by the schema. Like the low-level binding, high-level binding provides no facilities to cope with constant XML fragments. Therefore the formulation of constant XML fragments has

to be done by nested constructor or method calls, or by parsing of fixed documents, called unmarshalling. The first procedure is tedious for the programmer the second one needs validation at run-time. High-level bindings ensure well-formedness of dynamically generated documents at compile time. Static validity is only supported to a certain limited extent depending on the selected language mapping. Additionally they have been developed only for specific programming languages and are far away from becoming a standard.

With Validating DOM (VDOM), an extension of DOM, we introduced a high-level binding in previous work [KL01]. We coupled this binding with a mechanism called Parameterized XML (PXML) to support constant XML fragments guaranteeing static validity at compile time. The mechanism we use to guarantee the correctness in PXML is similar to an idea introduced in the setting of program generators about 20 years ago [Lin81]. The basic idea of that work was to introduce a data type for each nonterminal symbol of a context free grammar. So called generating expressions allow the program generator to insert values of these data types in places where the corresponding nonterminal symbol is allowed according to the underlying grammar. This mechanism guarantees the syntactical correctness of all generated programs statically.

Compile time validation

We are aware of only three approaches which are really comparable to XOBE.

The XDuce language [HVP00] is a functional language developed as an XML processing language. It introduces so called regular expression types the values of which are comparable to our XML objects. Elements are created by specific constructors and the content can be accessed through pattern matching. XDuce supports type inference for patterns and variables and performs a subtyping analysis to ensure validity of regular expression type instances at compile time.

BigWig [BMS01] is a programming language for developing interactive web services. It compiles BigWig source code into a combination of standard web technologies, like HTML, CGI, applets, and JavaScript. Typed XML document templates with gaps are introduced. In order to generate XML documents dynamically, gaps can be substituted at runtime by other templates or strings. For all templates BigWig validates all dynamically computed documents according to a given DTD. This is done by two data flow analyses constructing a graph which summarizes all possible documents. This graph is analyzed to determine validity of those documents. In comparison to our approach templates can be seen as methods returning XML objects. The arguments of the methods correspond to the gaps of the templates.

Another challenging approach is presented by the language specification of XL [FGK02]. XL is an XML programming language for the implementation of web services. In contrast to our approach, it is a stand-alone programming language, i.e. it is not an extension of an existing language like Java. It provides high-level and declarative constructs adopting XQuery. Additionally imperative language statements are introduced making XL a combination of an imperative and declarative programming language.

Additionally it is worth noticing that the upcoming standard of an XML query language XQuery [W3C02] has to support static validity as well.

Evaluation

In the following table we show how the different approaches give compile time guarantees and facilitate constant XML fragments.

	constant XML fragments	compile time guarantees well-formedness	static validity
CGI, Java Servlet	-	-	-
JSP	+	-	-
DOM, JDOM	-	+	-
JAXB, CASTOR	-	+	+
VDOM	+	+	+
XOBE, XDuce, BigWig, XL	+	+	++

The main observation of the table is that only XDuce, BigWig and XL are really comparable with our proposal. Other approaches either use only a string-based representation of XML structures or make a strict distinction between the string representation and the object representation of an XML structure.

XDuce implements a subtyping algorithm which is based on regular tree automata. It operates on an additional internal representation for regular expression types which is a source of inefficiency. This internal representation is avoided in our algorithm. Furthermore, because XOBE is an extension of Java, it is easier to couple it with other components like database systems.

BigWig's type checking algorithm is based on data flow analyses and is therefore totally different from our algorithm. Compared to XOBE we believe that our type system is more expressive because we can incorporate XML Schema's extension and restriction mechanisms quite naturally into the subtyping algorithm. This seems to be difficult in BigWig.

XL is defined as a special language for web services. In contrast, XOBE is defined as an extension of Java. Java is an already established programming language for web services. Thus XOBE can significantly benefit from Java by using already developed code.

8 Concluding Remarks

This paper presented the type checking algorithm of XOBE, an extension of the programming language Java. XOBE addresses the mismatch between Java on the one side and XML on the other by introducing XML objects which are defined by XML schema. It was shown that in XOBE the distinction between XML documents and XML objects no longer exists. XOBE is defined such that a running program works only with XML objects. XML documents in text form with explicit tagging are needed only for communicating with the outside world. The validity of all XML objects within a program can be checked at compile time by using XML constructors. In XML constructors, previously generated XML objects can be inserted in places which are allowed according to the un-

derlying XML schema. The type checking algorithm which was presented in this paper is used among others for checking the correctness of assignment statements involving XML objects. This correctness is checked by proving that the set of all XML objects which can be derived by evaluating the right hand side of an assignment is contained in the set of XML objects which are allowed as content of the variable on the left hand side according to the underlying XML schema. It was shown that well known notions from the literature can be enhanced for this purpose. In XOBE, the content of XML is accessed by XPath-expressions. A first prototype of the XOBE language extensions including the type system was implemented and some performance measures were provided.

In the future we plan to use XOBE in various application areas. One area will be the media archive software developed in Lübeck [Beh00]. The media archive implementation in its present state primarily uses Java Server Pages for generating HTML and XML. Other application areas will be looked at also. Moreover, we plan to integrate XQuery into the language. Allowing XML objects to be persistent results then in an XML-based database programming language.

Literaturverzeichnis

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web, From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, San Francisco, California, 2000.

[AG98] Ken Arnold and James Gosling. *The Java Programming Language*. The Java series. Addison Wesley Longman Limited, second edition, 1998.

[Ant94] Valentin Antimirov. Rewriting Regular Inequalities. In Reichel, editor, *Fundamentals of Computation Theory*, volume 965 of *LNCS*, pages 116–125. Springer Verlag Heidelberg, 1994.

[Beh00] Ralf Behrens. MONTANA: Towards a Web-based infrastructure to improve lecture and research in a university environment. In *Proceedings of the 2nd Int. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000), Milpitas, California*, pages 58–66. IEEE Computer Society, June 2000.

[BKMW01] Anne Brüggemann-Klein, Makoto Murata, and Derick Wood. Regular Tree and Regular Hedge Languages over Unranked Alphabets: Version 1. Technical Report HKUST-TCSC-2001-05, Hong Kong University of Science & Technology, April 3 2001. Theoretical Computer Science Center.

[BMS01] Claus Brabrand, Anders Moller, and Michael I. Schwartzbach. Static validation of dynamically generated HTML. In *Proceedings of Workshop on Program Analysis for Software Tools and Engineering (PASTE 2001), June 18-19, Snowbird, Utah, USA*. ACM, 2001.

[Bor01] Borland. *XML Application Developer's Guide, JBuilder*. Borland Software Corporation, Scotts Valley, CA, 1997,2001. Version 5.

[Bou02] Ronald Bourret. XML Data Binding Resources. web document, `http://www.rpbourret.com/xml/XMLDataBinding.htm`, 28. July 2002.

[Exo01] ExoLab Group. Castor. ExoLab Group, `http://castor.exolab.org/`, 11 December 2001.

[FGK02] Daniela Florescu, Andreas Grünhagen, and Donald Kossmann. XL: An XML Programming Language for Web Service Specification and Composition. In *Proceedings of*

International World Wide Web Conference (WWW 2002), May 7-11, Honolulu, Hawaii, USA. ACM, 2002. ISBN 1-880672-20-0.

[FK00] Duane K. Fields and Mark A. Kolb. *Web Development with Java Server Pages, A practical guide for designing and building dynamic web services*. Manning Publications Co., 32 Lafayette Place, Greenwich, CT 06830, 2000.

[Gai95] M. Gaither. *Foundations of WWW-Programming with HTML and CGI*. IDG-Books Worldwide Inc., Foster City, California, USA, 1995.

[HVP00] Haruo Hosoya, Jérôme Vouillon, and Benjamin C. Pierce. Regular Expression Types for XML. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada*, volume 35(9) of *SIGPLAN Notices*, pages 11–22. ACM, September 18-21 2000. ISBN 1-58113-202-6.

[JDO] JDOM Project. JDOM FAQ. `http://www.jdom.org/docs/faq.html`.

[Kem03] Martin Kempa. *Schema-abhängige Programmierung von XML-basierten Web-Anwendungen*. PhD thesis, Institut für Informationssysteme, Universität zu Lübeck, 2003. to appear, in german.

[KL01] Martin Kempa and Volker Linnemann. V-DOM and P-XML – Towards A Valid Programming Of XML-based Applications. In Akmal B. Chaudhri and Awais Rashid, editors, *OOPSLA '01 Workshop on Objects, XML and Databases, Tamba Bay, Florida, USA*, October 2001.

[KL02] Martin Kempa and Volker Linnemann. On XML Objects. In *Informal Proceedings of the Workshop on Programming Language Technologies for XML (PLAN-X 2002), PLI 2002, Pittsburgh, USA*, pages 44–54, 3.-8. October 2002.

[Kra02] Jens Kramer. Erzeugung garantiert gültiger Server-Seiten für Dokumente der Extensible Markup Language XML. Master's thesis, Institut für Informationssysteme, Universität zu Lübeck, 2002. in german.

[Lin81] Volker Linnemann. Context-free Grammars and Derivation Trees in Algol 68. In *Proceedings International Conference on ALGOL68, Amsterdam*, pages 167–182, 1981.

[LK02] Volker Linnemann and Martin Kempa. Sprachen und Werkzeuge zur Generierung von HTML- und XML-Dokumenten. *Informatik Spektrum*, 25(5):349–358, 2002. in german.

[Mic01] Microsoft Corporation. .NET Framework Developer's Guide. web document, `http://msdn.microsoft.com/library/default.asp`, 2001.

[Net97] Netscape Communications Corporation. JavaScript 1.1 Language Specification. `http://www.netscape.com/eng/javascript/index.html`, 1997.

[Ora01] Oracle Corporation. *Oracle9i, Application Developer's Guide - XML, Release 1 (9.0.1)*. Redwood City, CA 94065, USA, June 2001. Shelley Higgins, Part Number A88894-01.

[PLC99] Eduardo Pelegrí-Llopart and Larry Cable. Java Server Pages Specification, Version 1.1. Java Software, Sun Microsystems, `http://java.sun.com/products/jsp/download.html`, 30. November 1999.

[Sun01] Sun Microsystems, Inc. The Java Architecture for XML Binding, User Guide. `http://www.sun.com`, May 2001.

[The01] The Apache XML Project. Xerces Java Parser. `http://xml.apache.org/xerces-j/index.html`, 15. November 2001. Version 1.4.4.

[TWP00] Kevin Tao, Wanjun Wang, and Dr. Jens Palsberg. Java Tree Builder JTB. `http://www.cs.purdue.edu/jtb/`, 15. May 2000. Version 1.2.2.

[W3C98a] W3Consortium. Document Object Model (DOM) Level 1 Specification, Version 1.0. Recommendation, `http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/`, 1. October 1998.

[W3C98b] W3Consortium. Extensible Markup Language (XML) 1.0. Recommendation, http://www.w3.org/TR/1998/REC-xml-19980210/, 10. February 1998.

[W3C99] W3Consortium. XML Path Language (XPath), Version 1.0. Recommendation, http://www.w3.org/TR/xpath, 16. November 1999.

[W3C01a] W3Consortium. XML Schema: Formal Description. Working Draft, http://www.w3.org/TR/2001/WD-xmlschema-formal-20010925/, 25. September 2001.

[W3C01b] W3Consortium. XML Schema Part 0: Primer. Recommendation, http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/, 2. May 2001.

[W3C02] W3Consortium. XQuery 1.0: An XML Query Language. Working Draft, http://www.w3.org/TR/2002/WD-xquery-20021115/, 15. November 2002.

[Web02] WebGain. Java Compiler Compiler (JavaCC) - The Java Parser Generator. http://www.webgain.com/products/java_cc/, 2002. Version 2.1.

[Wil99] A. R. Williamson. *Java Servlets by Example*. Manning Publications Co., Greenwich, 1999.

[WS92] L. Wall and R. L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., Sebastipol, California, 1992.

A Shop Interchange Format

```
<schema>
   <element name="shopRequest" type="t_shopRequest"/>

   <complexType name="t_shopRequest"><sequence>
      <element name="shoppingCart" type="t_cartRequest"/>
   </sequence></complexType>

   <complexType name="t_cartRequest"><sequence>
      <element name="account" type="integer"/>
      <choice>
         <element name="add" type="integer"/>
         <element name="remove" type="integer"/>
         <element name="get"><complexType/></element>
      </choice>
   </sequence></complexType>

   <element name="shopResponse" type="t_shopResponse"/>

   <complexType name="t_shopResponse"><sequence>
      <element name="shoppingCart" type="t_cartResponse"/>
   </sequence></complexType>

   <complexType name="t_cartResponse"><sequence>
      <element name="account" type="integer"/>
      <element name="request" type="t_request"/>
      <element name="items" type="t_items" minOccurs="0" />
   </sequence></complexType>

   <complexType name="t_items"><sequence>
      <element name="article" type="integer" minOccurs="0" maxOccurs="
           unbounded"/>
      <element name="description" type="string" minOccurs="0"/>
   </sequence></complexType>

   <simpleType name="t_request"><restriction base="string">
      <enumeration value="processed"/>
      <enumeration value="fail"/>
   </restriction></simpleType>
</schema>
```

SQL/MM Spatial: The Standard to Manage Spatial Data in Relational Database Systems

Knut Stolze

Friedrich–Schiller–University Jena
Database and Information Systems Group
Ernst–Abbe–Platz 1–4
07743 Jena, Germany

stolze@informatik.uni-jena.de

IBM Entwicklung GmbH
DB2 Extenders Development
Schönaicher Str. 220
71032 Böblingen, Germany

stolze@de.ibm.com

Abstract: Several major database systems provide extensions to support the management and analysis of spatial data in a relational database system [IBM02, Ora01, IBM01]. The functionality is also standardized in ISO/IEC 13249 SQL/MM. This paper presents part 3 of the standard and discusses it critically. The spatial data types and methods on these types are explained. The Information Schema, showing spatial columns and spatial reference systems, is an important part for the handling of spatial data. It is described in the paper as well.

1 Introduction

ISO/IEC 13249 SQL/MM is the effort to standardize extensions for multi-media and application-speci c packages in SQL. SQL, as de ned in [ISO99], is extended to manage data like texts, still images, spatial data, or to perform data mining. The standard is grouped into several parts.

Part 1 is the framework for all the subsequent parts and de nes the de nitional mechanisms and conventions used in the other parts as well the common requirements that an implementation[1] has to adhere to if it wants to support any one of the extensions de ned in the standard. Part 2 is the full-text standard, which is concerned about the mechanisms to provide extended text search capabilities, above and beyond the operators provided by SQL, e. g. the LIKE predicate. Part 5 de nes the functionality to manage still images, and part 6 is concerned with data mining. The withdrawn part 4 addressed general purpose facilities.

ISO/IEC 13249-3 SQL/MM Part 3: Spatial [ISO02c] is the international standard that de nes how to store, retrieve and process spatial data using SQL. It de nes how spatial data is to be represented as values, and which functions are available to convert, compare, and process this data in various ways.

[1] The SQL standards use the term *implementation* to refer to a program that implements the interfaces defined by the standard. Commonly, an implementation is a relational database system.

The first version of the standard was published in 1999. In the years since then, several enhancements were added to the document, and the second version is now available as Final Draft International Standard (FDIS). It is expected that it will be published as International Standard (IS) in the near future.

In this paper, geometries like points, lines, and polygons or composites thereof are also referred to as *spatial data*. Geometries can be used in many different application domains. The model as specified by the SQL/MM standard is applicable to a variety of different data spaces. The spatial reference system associated for a geometry identifies the data space used for that geometry. For example, it defines whether a geometry models a geographic feature or some other, more abstract feature.

The most common case where spatial data is used in practice are geographic information systems (GIS). There, the term *geometry* is used to denote the geometric features that cartographers have used for the past centuries to map the world. An abstract definition of the meaning of geographic geometry is a point or aggregate of points representing a feature on the ground. Thus, a geometry is usually a model of a geographic feature. The model can be expressed in terms of the feature's coordinates. The model conveys information; for example, the coordinates identify the position of the feature with respect to fixed points of reference.

In the non-geographic applications, a geometry can identify, for example, a feature in a still image where no relation to the earth can be established. Another example are locations inside a grocery store. Although a relation to the earth could be computed based on the latitudes and longitudes, a preferred representation for an application might only refer to the locations with respect to a fixed point in the store, e. g. the south-east corner.

The SQL/MM standard is divided into clauses. The clauses 5 thru 9 describe the geometry types and the methods provided for each type. The Information Schema, based on a Definition Schema is defined in clause 14. The remaining clauses, which are not described in this paper, explain the underlying spatial concepts, the angles and direction handling, and the States codes and conformance rules for products that implement the standard.

Several products exist, which implement spatial extensions for relational database systems. For example, the DB2 Spatial Extender is available for IBM's DB2 Universial Database (UDB), the IDS Spatial DataBlade and Geodetic DataBlade for IBM's Informix Dynamic Server (IDS), and Oracle offers the Oracle 9i Spatial product. At should be noted that the list given here only shows the more commonly known products and additional products can be found as well.

The paper explains and discusses the content of the main clauses in the standard in more detail. Section 2 shows which spatial data types exist, how they are organized, and which functionality is provided for each type. Two short examples show how to use the spatial functionality in a relational database in section 3. Like any other part of SQL/MM, the Spatial standard contains an Information Schema. The views in that Information Schema are listed in section 4. The paper is completed with a summary and outlook on the possible future developments of the standard in section 5.

2 Spatial Data Types

2.1 History

The roots of the SQL/MM Spatial standard are directly apparent in the type hierarchy. The standard was originally derived from the OpenGIS Simple Features Speci cation for SQL [OGC99], also published in the year 1999 as version 1.1 by the OpenGIS Consortium (OGC). The Simple Feature Speci cation de nes a so-called Geometry Model . The geometry model consists of a class hierarchy, which is shown in gure 1.

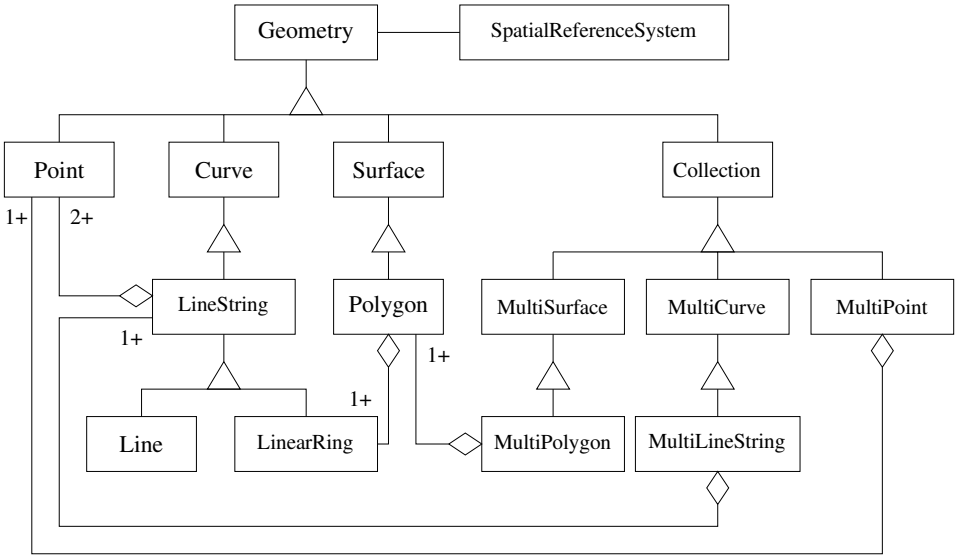


Figure 1: OpenGIS Geometry Class Hierarchy

The geometry model is an abstract model. It is used to de ne the relationship between the various classes and to establish the inheritance rules for the methods working on the instances of the classes and subclasses. For example, the method *Area* is de ned for the class *Surface* and is available for all instances of *Surface*, *Polygon*, and further subclasses, whereas the method *ExteriorRing* is only de ned on the subclass *Polygon* and, thus, cannot be used for arbitrary instances of the class *Surface*.

The SQL/MM standard uses consistently the pre x `ST_` for all tables, views, types, methods, and function names. The pre x stood originally for *Spatial* and *Temporal*. It was intended in the early stages of the standard development to de ne a combination of temporal and spatial extension. A reason for that was that spatial information is very often tied with temporal data [SWCD98, SWCD97, RA01, TJS97]. During the development of SQL/MM Spatial, it was decided that temporal has a broader scope beyond the spatial application and should be a part of the SQL standard [ISO99] as SQL/Temporal [ISO01]. The con-

tributors to SQL/MM did not want to move forward with a Spatio-temporal support until SQL/Temporal developed.[2] In the mean time, thefocus of spatial standard lied on keeping it aligned with the OGC speci cation and the standards developed by the technical commitee ISO/TC 211, for example [ISO02a, ISO02b]. The pre x `ST_` for the spatial tables, types, and methods was not changed during the organizational changes of the standards, however. Today, one might want to interpret it as *Spatial Type*.

2.2 Geometry Type Hierarchy

The OGC geometry class hierarchy is adapted for the corresponding SQL type hierarchy that is de ned in the SQL/MM standard. Figure 2 shows the standardized type hierarchy. The shaded types are the not-instantiable types.[3] All types are used to represent geometric features in the 2-dimensional space ($\mathbf{R}^2$).

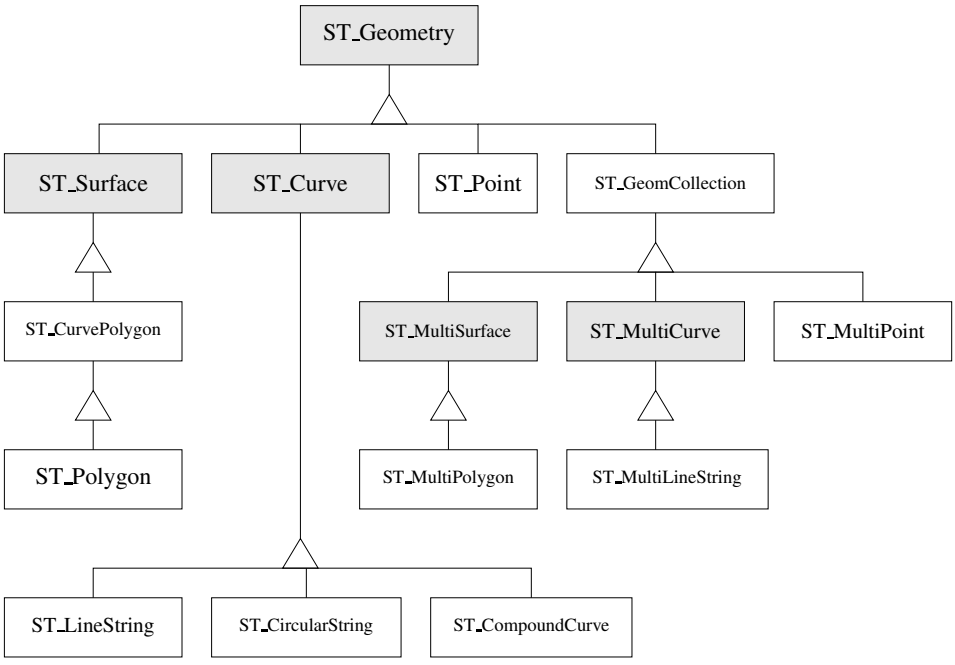


Figure 2: SQL Type Hierarchy

The major differences between the SQL type hierarchy and the OGC geometry class hierarchy are the omission of the derived types *Line* and *LinearRing*, and the addition

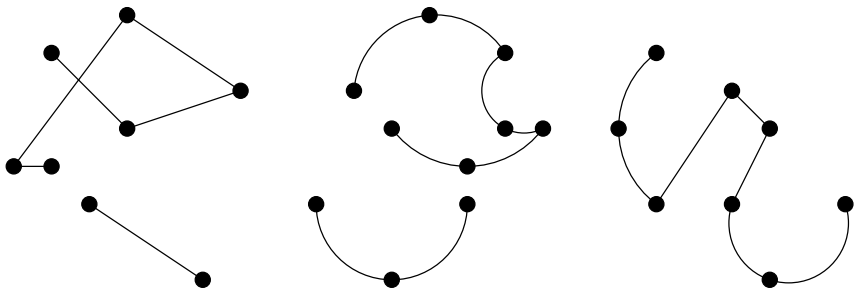
[2] SQL/Temporal was not any further developed and, like SQL/MM Part, subsequently withdrawn completely.

[3] It is implementation-defined whether *ST_MultiCurve* and *ST_MultiSurface* are instantiable or not, even though they are shown as not-instantiable in figure 2.

of a series of types. Lines and linear rings are to be represented using values of type *ST_LineString*, which covers both cases. The new types extend the OGC geometry class hierarchy with circular arcs as curves and surfaces that have circular arcs as their boundary. Furthermore, the aggregations that re ect which types are used by other types are not shown. For example, it is not obvious from the SQL type hierarchy that the type *ST_MultiPoint* consists of *ST_Point* values.

ST_Point values are 0-dimensional geometries and represent only a single location. Each point consists of an X and a Y coordinate to identify the location in the respective spatial reference system. Points can be used to model small real-world objects like lamp posts or wells. *ST_MultiPoint* values stand for a collection of single points. The points in a multi-point do not necessarily have to be distinct points. That means a multi-point supports sets as in the strict mathematical sense, but also allows for multi-set like SQL does in general.

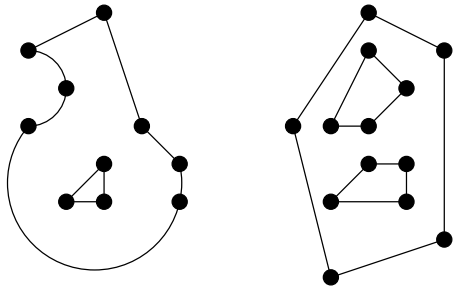
Curves are 1-dimensional geometries. The standard distingiushes between *ST_LineString*, *ST_CircularString*, and *ST_CompoundCurve*. An *ST_LineString* is de ned by a sequence of points, (X, Y) pairs, which de ne the reference points of the line string. Linear interpolation between the reference points de nes the resulting linestring. That means, two consecutive points de ne a line segment in the linear string. Circular instead of linear interpolation is used for *ST_CircularString* values. Each circular arc segment consists of three points. The rst point de nes the start point of the arc, the second is any point on the arc, other than the start or end point, and the third point is the end point of the arc. If there is more than one arc in the circular string, the end point of one arc acts as the start point of the next arc. A combination of linear and circular strings can be modeled using the *ST_CompoundCurve* type. Line segments and circular segments can be concatenated into a single curve. *ST_MultiCurve* values represent a multi-set of *ST_Curve*, and *ST_MultiLineString* a multi-set of *ST_LineString* values. Note that there are no types *ST_MultiCircularString* and *ST_MultiCompoundString*. Figure 3 illustrates some examples of the three different types of curves.



(a) Linear Strings (b) Circular Strings (c) Compounds

Figure 3: Examples of Curves

Surfaces, as 2-dimensional geometries, are de ned in the same way as curves using a sequence of points. The boundary of each surface is a curve, or a set of curves if the surface has any holes in it. The boundary of a surface consists of a set of rings, where each ring is a curve. The type *ST_CurvePolygon* stands for such a generalized surface, and the subtype *ST_Polygon* restricts the conditions for the rings of the boundary to linear strings. The types *ST_MultiSurface* and *ST_MultiPolygon* are used to model sets of curve polygons or polygons with linear boundaries.[4] A curve polygon and a polygon with linear strings as its boundary are shown in gure 4.



(a) Curve Polygon (b) Polygon

Figure 4: Examples of Polygons

The type hierarchy does not address the concept of empty geometries in form of a separate type under *ST_Geometry*. An empty geometry represent an empty set of points, and it can be the result of the intersection of two disjoint polygons. The standard allows that a value of each of the instantiable types can be an empty geometry. Thus, empty points, empty linestrings, empty polygons, and empty geometry collections etc. exist. If a method has the return an empty geometry as its result and the most speci c type is not inherent by the method, then an empty point is generated. Implicit or explicit casts can be used to convert empty geometries from one type to another.

2.3 Methods

The majority of all spatial methods can be grouped into one of the following four categories:

- convert between geometries and external data formats,
- retrieve properties or measures from a geometry,
- compare two geometries with respect to their spatial relation,

[4] *ST_MultiSurface* constrains its values to contain only disjoint surfaces.

- generate new geometries from others

Examples and descriptions for each of the categories are given in this section. Sometimes, it is not trivial to assign a spatial method to only a single category. For instance, the method *ST_StartPoint*, which returns the first point of a linestring, retrieves a property of the linestring, i. e. the first point, but it also generates a new geometry, i. e. an *ST_Point* value.

2.3.1 Convert to and from External Data Formats

The SQL/MM standard defines three external data formats that can be used to represent geometries in an implementation-independent fashion.

- well-known text representation (WKT)
- well-known binary representation (WKB)
- geography markup language (GML)

Each type implements constructor methods that allow to generate a new geometry from the given WKT or WKB and the, optionally, provided numeric spatial reference system identifier. All instantiable types have such constructor methods. There are no constructor methods that cope with the GML representation. Functions like *ST_LineFromGML* or *ST_MPointFromGML* are used instead.

For backward compatibility, the standard also defines functions like *ST_PointFromText* or *ST_GeometryFromWKB* with exactly the same purpose as the constructor methods. Those functions were inherited from and remain for compatibility with the OpenGIS Simple Feature Specification for SQL [OGC99]. The constructor methods were introduced later in the process of the standard development to align part 3 of SQL/MM with other parts, and also to improve the overall usability of the constructors.

The three methods *ST_AsText*, *ST_AsBinary*, and *ST_AsGML* are provided for the conversion of a geometry to the respective external data format.

2.3.2 Retrieve Properties

All geometries have certain properties. A property is, for example, the dimension or if a geometry is empty. Each of the subtypes adds further, more specific properties, for example, the area of a polygon or whether a curve is simple[5]. A set of method was defined to query those properties. Due to the high number of available methods, only a small set of examples is given here.

ST_Boundary return the boundary of a geometry

[5] A *simple* curve is defined to be not self-intersecting.

ST_IsValid test whether a geometry is valid, i. e. correctly de ned; an invalid geometry could be a not-closed polygon

ST_IsEmpty test whether a geometry is empty

ST_X return the X coordinate of a point

ST_IsRing test whether a curve is a ring, i. e. the curve is closed and simple

ST_Length return the length for a linestring or multi-linestring

2.3.3 Compare Two Geometries

A very interesting part in spatial queries comes from the comparison of geometries. Questions like which buildings are in a ood zone or where are intersections of rail roads and streets can only be answered if the geometries representing the buildings, ood zones, rail roads, and streets are compared with each other.

The standard de nes the following set of methods to compare geometries in various ways.

ST_Equals test the spatial equality of two geometry

ST_Disjoint test whether two geometries do not intersect

ST_Intersects**, *****ST_Crosses*****, and *****ST_Overlaps*** test whether the interiors of the geometries intersect

ST_Touches test whether two geometries touch at their boundaries, but do not intersect in their interiors

ST_Within **and** ***ST_Contains*** test whether one geometry is fully within the other

All of the above methods return an INTEGER value, which is 1 (one) if the spatial relation does exist, and 0 (zero) otherwise.

Additionally, the method *ST_Distance* exists, which quanti es the spatial relationship of two geometries according to their distance.

2.3.4 Generate New Geometries

The last set of methods allows the user to generate new geometries from existing ones. A newly generated geometry can be the result of a set operation on the set of points represented by each geometry, or it can be calculated by some algorithm applied to a single geometry. The following methods are examples for both.

ST_Buffer generate a buffer at a speci c distance around the given geometry

ST_ConvexHull compute the convex hull for a geometry

ST_Difference**, *****ST_Intersection*****, and *****ST_Union*** construct the difference, intersection, or union between the point sets de ned by two geometries

2.4 Discussion

An observation of the type hierarchy as de ned by the SQL/MM standard raises several questions on the design decisions that were made.

2.4.1 OGC Geometry Model

The OGC geometry class hierarchy attempts to implement the *composite* design pattern [GHJV95] by adding the class *Collection*. Without the subclasses under *Collection* and an additional aggregation of *Geometry*, the pattern would t exactly. A simpli cation of the type hierarchy could have been achieved with the omission of all the subclasses of *Collection* without any loss of functionality. All the methods that are de ned on the subclasses of *Collection* have the same simple logic. The same method is called for each element (also called *part*) of the collection, and the results are combined.

Another drawback of this way of modeling the classes is that a future extension of the geometry model requires the handling of the new geometries in two different places. First, the new class has to be added under *Geometry*, and second, a corresponding class has to be added for sets of such a geometry under *Collection*.

2.4.2 SQL Type Hierarchy

A different picture shows the implementation of the OGC geometry class hierarchy as the SQL/MM type hierarchy. SQL is a set-oriented language. An important goal of a standard should be the usability of the functionality de ned. Iterating over the elements of a collection as discussed above is, although possible, not as simple in SQL. A dynamic compound statement or a recursive query has to be used for the iteration. The respective method on the element of a collection has to be invoked during the iteration, and the results are to be combined. Leaving that task up to the user will only lead to the user de ning those functions himself to prevent the repetition of that task.

Furthermore, consider the following typical user scenario: The SQL query is supposed to return all the parts of the geometries in column `spatial_column` that fall into a certain rectangle. The rectangle in question is represented using a polygon.

```
SELECT ST_Intersection(ST_Polygon(
            'polygon((10 10, 10 20, 20 20, 20 10,
                      10 10)), 1), spatial_column)
FROM   spatial_table
WHERE  ...
```

Assuming that the column contains values of type *ST_MultiPoint*, the results of the query can contain any of the following for each row:

- an empty geometry (empty point)

- a single point
- multi-points

In the above query, the results can be retrieved and visualized on the screen. Further processing the results in an SQL statement is, however, not completely trivial. Because *ST_Point* and *ST_MultiPoint* are in two independent subtrees of the type hierarchy, only the methods available on the common ancestor *ST_Geometry* can be used. A conversion of the single points to multi-points with only a single element is not supported by the standard. It only supports the conversion of empty geometries from one type to another.

A solution to address this issue might be to set the types *ST_Point* and *ST_MultiPoint* in direct relationship by using inheritance. Interpreting that in the context of the SQL type hierarchy gives: *ST_MultiPoint* is a specialized *ST_GeomCollection*, which only contains points, and *ST_Point* is a specialized *ST_MultiPoint*, consisting of only a single point. A similary approach can be implemented for the other types, which are not in the subtree under *ST_GeomCollection*. Figure 5 shows how such a type hierarchy could be de ned.

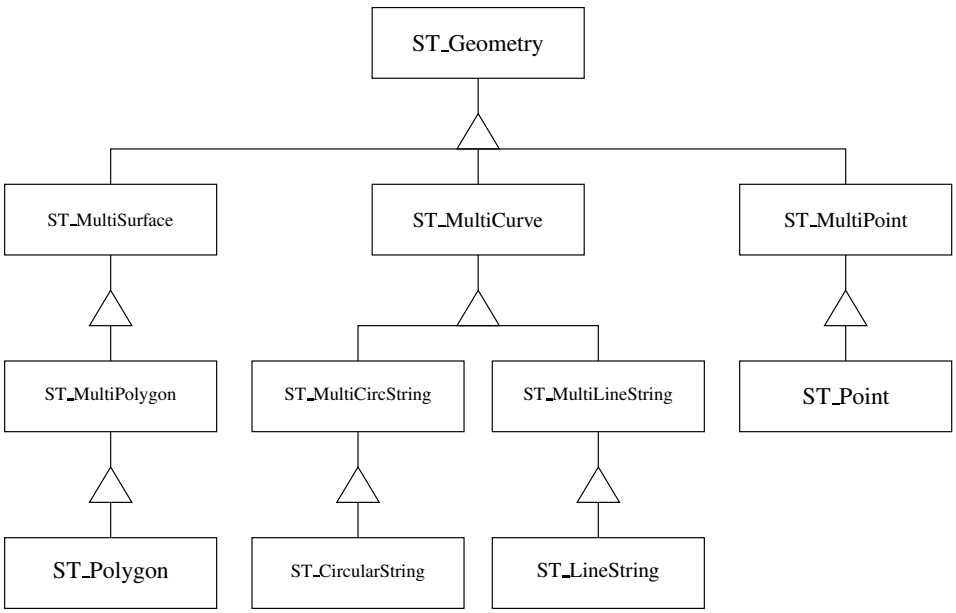


Figure 5: Modified Type Hierarchy

A similar idea was apparently the bases for the type hierarchy implemented in the SpatialWare DataBlade for IDS product [Map02]. The DB2 Spatial Extender de nes a type hierarchy exactly as in the standard, but it omits some of the optional data types [IBM02]. And yet a third approach to handle geometries stems from the Oracle 9*i* Spatial product [Ora01]. It does not attempt to model a type hierarchy to re ect more speci c properties of the different kinds of geometries but uses a single type *SDO_Geometry* instead. No

support for strong typing based on the geometries can be enforced in such an environment, but other means have to be used.

2.4.3 Methods on the Geometry Types

The SQL/MM standard provides a rich set of methods and functions. But it can be noted that some additions and changes would result in a further improvement.

The standard de nes constructor methods that can handle well-known text (WKT) and well-known binary (WKB) representations. It does not allow for a handling of the GML representation in a constructor, however. The existing constructor methods for the well-known text representation could be reused for that, given that WKT and GML are both a textual representation for geometries and can easily be distinguished be analyzing the very rst non-whitespace character. The functions like *ST_PolyFromGML* could then be removed. They are not de ned in the OGC Simple Feature speci cation, so that compatibility issues do not arise.

A set of functions allows the user to construct any geometry using one of the external data formats. Those functions act like factory functions and are named *ST_GeomFromText*, *ST_GeomFromWKB*, and *ST_GeomFromGML*. Instead of using the explicit names to denote the format handled by each function, an approach similar to the constructor methods is preferrable. An overloaded function *ST_Geometry* can be de ned with the same semantical behaviour as the existing functions. Note that constructor methods on the type *ST_Geometry* cannot be used because that type is not instantiable.

There are several groups of methods that provide (nearly) identical functionality. For example, the methods *ST_Intersects*, *ST_Crosses*, and *ST_Overlaps* all test for intersections of the interiors of the two input geometries. The only difference between the methods is that *ST_Crosses* does not allow to test if a surface intersects some other geometry, or if some other geometry intersects a point. *ST_Overlaps* requires that both geometries to be compared have the same dimension. For example, a line can overlap another line but not a polygon. *ST_Intersects* is the generalized version of the functionality to test for the overlay of geometries. It does not impose any restrictions in its input parameters. The existence of *ST_Crosses* and *ST_Overlaps* is rather questionable.

Another omission can be found in the support for external data formats. The de-facto industry standard to represent geometries is the so-called shape format [ESR97]. The shape format is not supported by the standard, but it should be considered given that existing major products already support it [IBM02, IBM01].

3 User scenarios

Two user scenarios for spatial functionality as de ned in the standard as shown in this section. The rst scenario given in section 3.1. The second scenario describes how a bank can manage its customers and make decisions on the placement of new branches can be found in section 3.2.

3.1 Insurance Company

After a recent ooding, an insurance company wants to correct the information about insured buildings that are in the ood zone, and pose an increased risk for the company. The database contains one table `rivers` that contains the rivers and their ood zones and another table `buildings` with the data for the buildings of all the policy holders.

```
rivers(name, water_amount, river_line, flood_zones)
buildings(customer_name, street, city, zip, ground_plot)
```

The column `river_line` contain the linstrings that represent all the rivers in the country. Related to that the column `flood_zones` shows the oodzones for each river. The ground plot of the customer's building is stored in the column `ground_plot`. The tables for the above shown relational model can be created with the following SQL statements.

```
CREATE TABLE rivers (
   name          VARCHAR(30)  PRIMARY KEY,
   water_amount  DOUBLE PRECISION,
   river_line    ST_LineString,
   flood_zones   ST_MultiPolygon )

CREATE TABLE buildings (
   customer_name   VARCHAR(50)  PRIMARY KEY,
   street          VARCHAR(50),
   city            VARCHAR(20),
   zip             VARCHAR(10),
   ground_plot     ST_Polygon )
```

The rst task is to update the information about the ood zones. The ood zones for the river FLOOD is to be extended by 2 kilometers in each direction. The method *ST_Buffer* is used in the following SQL statement to extend the ood zones by the speci ed radius.

```
UPDATE rivers
SET    flood_zones =
          flood_zones.ST_Buffer(2, 'KILOMETER')
WHERE  name = 'FLOOD'
```

In the next step, the company wants to nd all the customers that are now in the extended ood zone for the river. An SQL statement involving the spatial method *ST_Overlaps* can be used to to nd all those buildings.

```
SELECT customer_name, street, city, zip
FROM   buildings AS b, rivers AS r
WHERE  b.ground_plot.ST_Within(r.flood_zones) = 1
```

The so retrieved addresses can be further processed, and the customers can be informed of any changes to their policy or other information.

3.2 Banking

A bank manages its customers and branches. Each customer can have one or more accounts, and each account is managed by a branch of the bank. To improve the quality of services, the bank performs an analysis of its customers, which also involves a spatial component, the locations of the customer's homes and the branches. The tables in the bank's data base have the following de nition.

```
CREATE TABLE customers (
   customer_id  INTEGER
      PRIMARY KEY,
   name         VARCHAR(20),
   street       VARCHAR(25),
   city         VARCHAR(10),
   state        VARCHAR(2),
   zip          VARCHAR(5),
   type         VARCHAR(10),
   location     ST_Point);
```

```
CREATE TABLE branches (
   branch_id  INTEGER
      PRIMARY KEY,
   name       VARCHAR(12),
   manager    VARCHAR(20),
   street     VARCHAR(20),
   city       VARCHAR(10),
   state      VARCHAR(2),
   zip        VARCHAR(5),
   location   ST_Point,
   zone       ST_Polygon);
```

```
CREATE TABLE accounts (
   account_id   INTEGER  PRIMARY KEY,
   routing_no   INTEGER  NOT NULL,
   customer_id  INTEGER  NOT NULL,
   branch_id    INTEGER  NOT NULL,
   type         VARCHAR(10)  NOT NULL,
   balance      DECIMAL(14, 2)  NOT NULL,
   CONSTRAINT fk_customers FOREIGN KEY(customer_id)
      REFERENCES customers(customer_id),
   CONSTRAINT fk_branches FOREIGN KEY(branch_id)
      REFERENCES branches(branch_id) );
```

The rst query determines all customers with an account balance larger than $10,000.- in any of the accounts and who live more than 20 miles away from their branch.

```
SELECT DISTINCT c.customer_id, c.name
FROM   customers AS c JOIN accounts AS a ON
          ( c.customer_id = a.customer_id )
WHERE  a.balance > 10000 AND
       a.location.ST_Distance(
          ( SELECT b.location
            FROM   branches
            WHERE  b.branch_id = a.branch_id ),
          'MILES') > 20
```

The bank wants to find all the portions of the assigned sales zones of the branches that overlap. It is not intended to have more than one branch assigned to a certain area, and any duplicates are to be found and corrected. The query retrieves the identifiers for each two branches that have an overlap in the zones and also the overlapping area, encoded in the well-known text representation.

```
SELECT b1.branch_id, b2.branch_id,
       b1.zone.ST_Overlaps(b2.zone).ST_AsText()
FROM   branches AS b1 JOIN branches AS b2 ON
          ( b1.branch_id < b2.branch_id )
WHERE  b1.zone.ST_Overlaps(b2.zone).ST_IsEmpty() = 0
```

To reduce competition amoungst the branches for its own customers, the bank wants to find all the customers that live within a 10 mile radius of a branch, which does not manage their accounts. The accounts are to be transferred to a closer branch if the customer agrees.

```
SELECT c.name, c.phone, b.branch_id
FROM   branches AS b, customers AS c
WHERE  b.location.ST_Buffer(10, 'MILES').
          ST_Contains(c.location) = 1 AND
       NOT EXISTS (
          SELECT 1
          FROM   accounts AS a
          WHERE  a.customer_id = c.customer_id AND
                 a.branch_id = b.branch_id )
```

4 Information Schema

SQL/MM Part 3: Spatial defines an Information Schema to provide an application that uses the spatial extension with a mechanism to determine the supported and available features. The Information Schema consists of four views, which are explained here, after a short introduction on the history.

4.1 History

The SQL/MM Information Schema was also inherited from the OGC Simple Feature Specification for SQL [OGC99]. The OGC specification used ad still uses the views `GEOMETRY_COLUMNS` and `SPATIAL_REF_SYS` with a different set of columns and different semantics of the data shown in the view.

The OGC specification describes two completely different concepts, called environments, that can be used for the implementation of a spatial extension for a database system, based on whether structured types, so-called ADTs, are supported or not. The Information

Schema for the environment without structured types is more complex because additional information has to be maintained.

For the second environment that exploits the structured type support, which is in alignment with the SQL/MM standard, the OGC specification refers in the description for the `GEOMETRY_COLUMNS` view only back to the other environment and states: the columns in the GEOMETRY_COLUMNS metadata view for the SQL92 with Geometry Types environment are a subset of the columns in the GEOMETRY_COLUMNS view defined for the SQL92 environment. It is not exactly clear what the subset is supposed to be.

The original Information Schema defined in the SQL/MM standard was based on those information. For example, the view `GEOMETRY_COLUMNS` included a column named `COORD_DIMENSION`, which has no well-defined meaning when storing arbitrary geometries on a spatial column. The original views were replaced by a new definition of the Information Schema in the second edition of the SQL/MM standard. The following sections refer to the newly introduced views.

4.2 SQL/MM Spatial Information Schema

The Spatial Information Schema consists of 4 views that list the spatial columns, the supported spatial reference systems, the units of measure, and the implementation-defined meta-variables. The entity-relation-ship diagram in figure 6 shows those views and also their relationship to the views defined in the Information Schema in [ISO99].

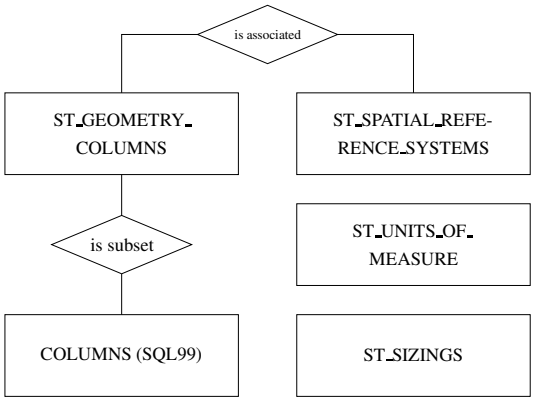


Figure 6: Spatial Information Schema

ST_GEOMETRY_COLUMNS The view lists all columns in all tables that have a declared type of *ST_Geometry* or one of its subtypes. It is not necessary to associate a specific spatial reference system with a column, but an application can do so. The view shows the

column identifier, consisting of catalog, schema, table, and column name, and the identifying name and the numeric identifier of the spatial reference system associated with the column.

The view is written in such a way to query the view `COLUMNS` from the SQL Information Schema to retrieve the infermation about all existing spatial columns and then merges the SRS information for each of the columns that has an associated spatial reference system using an outer join.

ST_SPATIAL_REFERENCE_SYSTEMS A spatial reference system has two unique identifiers, a name and a numeric identifier. The name is used in the same way as for all other SQL objects like schemata, functions, or columns. The numeric identifier is used in the methods that require the SRS information as input, for instance to construct a geometry in a specific SRS or to transform a geometry to another SRS.

Along with the identifiers, the view represents the organization that defined this spatial reference system together with the identifier assigned by that organization and the actual definition of the SRS.

ST_UNITS_OF_MEASURE Different units can be used to calculate distances between geometries, the length of curves, or the area of surfaces. The view lists those units that are supported. An identifying name, the type of the unit (angular or linear), and the conversion factor to the base unit within each type is stored. The conversion factor for the base units in each type are always 1 (one).

ST_SIZINGS Similar to the SQL99 Information Schema view `SIZINGS`, the SQL/MM standard requires that an implementation creates a view `ST_SIZINGS`. This view contains the spatial-specific meta-variables and their values. An example of a meta-variable is the maximum possible length that can be used for a well-known text representation of a geometry. This meta-variable is called *ST_MaxGeometryAsText*.

4.3 Discussion

Two views in the Information Schema should be reconsidered because their current definition is not adequate. First, the view `ST_SPATIAL_REFERENCE_SYSTEMS` uses a very simplified way to manage spatial reference systems. The European Petrol Survey Group (EPSG) worked on a more expressive schema for spatial reference systems, which is also described in another ISO standard [ISO02b], although in a different context.

The view `ST_SIZINGS` has the same intention as the view `SIZINGS` defined in SQL99 [ISO99], only specialized for the facilities in SQL/MM Spatial. If SQL99 would provide a mechanism for implementations of other standards, including SQL/MM to add new entries to its view, then `ST_SIZINGS` becomes obsolete and could be removed from the SQL/MM

standard. Unfortunately, SQL99 does not describe the requested mechanisms, so both views are still necessary.

5 Summary and Outlook

SQL/MM Part 3: Spatial standardizes the storing, retrieving and processing of spatial data as part of a relational database system. It de nes a set of types and methods for the representation of 0, 1, or 2-dimensional geographic features.

The SQL/MM standard is expected to be published in its second version in the near future. The second version shows improvements for the Information Schema and adds the support for the Geography Markup Language (GML) and angles and directions. It also de nes many functions in a more precise manner.

This paper described the facilities de ned in the standard, and discussed them critically. The implemented type hierarchy is suitable for its purpose, but together with the strong typing and the set-oriented data management imposed be SQL, it inconveniences the spatial data processing. The methods provided for each type cover a wide range of spatial functionality. The multitude of methods lead to the situation where functionality is already duplicated (with very minor differences).

Some future directions of the SQL/MM Spatial standard already show in the new working draft that was started recently and will eventually become the third version of the standard. The Japanese standards committee supplied a change proposal that adds a function *ST_ShortestPath*, which calculates the shortest path in a network (or graph) of linestrings between two given points.

Additional functionality that implements more spatial oriented logic could be included as well in the future. For example, a function *ST_Nearest* to nd for a given geometry the nearest one from another set of geometries is desirable. The user cousd exploit it to get the answers to questions like nd me the closest restaurant to my current location .

The support for modi cation of geometries directly in the database system using spatial methods can be extended. There are no simple functions to change a point in a linestring, or to generalize geometries if it is too detailed, i. e. too many points are used to de ne it. Existing products already support methods like *SE_ChangeVertex* or *SE_Generalize* [IBM01].

Open questions also remain with respect to other SQL standards. For example, the Information Schema de ned in the SQL/MM Spatial standard contains information about the spatial reference systems applicable for geometries stored in the same database as the Information Schema. The access to external data stores using SQL/MED [ISO00] is not considered today.

The support for raster data, for example huge images taken for whole countries, imposes special requirements on a spatial database. Geographical Information Systems (GIS) rely on raster data to provide additional information for the user. The current SQL/MM standard does not consider raster data at all, and the needed infrastructure should be de ned.

Literaturverzeichnis

[ESR97] Environmental Systems Research Institute, Inc. *ESRI Shapefile Technical Description*, 1997.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissidis. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[IBM01] International Business Machines, Corp. *Informix Spatial DataBlade, Version 8.11*, 2001.

[IBM02] International Business Machines, Corp. *DB2 Spatial Extender – User's Guide and Reference, Version 8.1*, 2002.

[ISO99] ISO/IEC 9075-2:1999. *Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation)*, 1999.

[ISO00] ISO/IEC 9075-9:2000. *Information Technology – Database Languages – SQL – Part 9: SQL/MED*, 2000.

[ISO01] ISO/IEC 9075-2:2001 WD. *Information Technology – Database Languages – SQL – Part 7: Temporal (SQL/Foundation)*, 2001.

[ISO02a] ISO/DIS 19107:2002. *Geographic Information - Spatial Schema*, 2002.

[ISO02b] ISO/DIS 19111:2002. *Geographic Information - Spatial Referencing by Coordinates*, 2002.

[ISO02c] ISO/IEC 13249-3:2002 FDIS. *Information technology – Database languages – SQL Multimedia and Application Packages – Part 3: Spatial*, 2nd edition, 2002.

[Map02] MapInfo, Corp. *MapInfo SpatialWare – User's Guide, Version 4.5*, 2002.

[OGC99] OpenGIS Consortium. *OpenGIS Simple Features Specification for SQL, Revision 1.1*, 1999.

[Ora01] Oracle, Corp. *Oracle Spatial User's Guide and Reference, Release 9.0.1*, 2001.

[RA01] K. H. Ryu and Y. A. Ahn. Application of Moving Objects and Spatiotemporal Reasoning. Technical report, TimeCenter, 2001.

[SWCD97] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE13)*, Birmingham, UK, 1997.

[SWCD98] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. *Temporal Databases: Research and Practice*, chapter Querying the Uncertain Position of Moving Objects. Springer-Verlage, 1998.

[TJS97] V. J. Tsotras, C. S. Jensen, and R. T. Snodgrass. A Notation for Spatiotemporal Queries. Technical report, TimeCenter, 1997.

Verbalisierung von Datenbanktransaktionen

Olaf Th. Buck
Fa. Peter Pietsch, Organisationsberatung und Informationstechnologie
Grosse Burgstrasse 55-57
D-23552 Lübeck
Email: olaf.buck@pietsch-luebeck.de
http://www.pietsch-luebeck.de

Volker Linnemann
Institut für Informationssysteme
Universität zu Lübeck
Osterweide 8
D-23562 Lübeck
Email: linnemann@ifis.uni-luebeck.de
http://www.ifis.uni-luebeck.de

Abstract: Datenbanktransaktionen bestehen in der Regel aus mehreren Datenbankanweisungen und haben vielfältige Veränderungen des Datenbestandes zur Folge. Diese Veränderungen sind in der Regel für einen Anwender, der die Details der Transaktion nicht kennt, nicht direkt nachvollziehbar. Daher ist es wünschenswert, wenn dem Anwender eine natürlichsprachliche Erklärung der Auswirkung an die Hand gegeben werden kann, mit Hilfe derer er die Transaktion auf einem hohen Abstraktionsniveau nachvollziehen kann. Darüber hinaus ist es zur Erfüllung von Rechtsvorschriften, wie zum Beispiel für die Sicherstellung von Revisionssicherheit, notwendig, Strategien und Funktionen zur Verbalisierung von Datenbanktransaktionen zu entwickeln. Es wird in dieser Arbeit gezeigt, wie Transaktionen in einem geeigneten Format aufgezeichnet und mit Hilfe von Schablonen in natürliche Sprache übersetzt werden können. Abschließend wird ein System zur Realisierung dieser Funktionen vom Entwurf bis zur Implementierung beschrieben. Dieses wird zur Zeit bei mehreren Anwendern erfolgreich eingesetzt.

1 Einleitung

Benutzer eines Systems mit Datenbankunterstützung navigieren selten direkt auf der Datenbank durch Verwendung einer Anfragesprache wie SQL. In der Regel ist eine Applikation mit statischen Eingabemasken oder ähnlichem vorhanden, in deren Quellcode die entsprechenden Anfragen gekapselt sind. So werden Änderungen in der Datenbank und deren Auswirkungen für Benutzer des Systems wenig transparent. Dies wird auch dadurch verstärkt, dass im Allgemeinen Benutzer nicht exklusiv mit einer Datenbank arbeiten, sondern diese mehreren Benutzern gleichzeitig zur Verfügung steht. Darüber hinaus sind Datenbankschemata selten statisch, sondern sie verhalten sich gemäß des Softwarelebenszyklus von darauf operierenden Applikationen dynamisch, wobei sich das Aussehen beispielsweise von Eingabemasken nicht zwingend verändern muss. Veränderungen des Schemas können nicht nur durch veränderte Anforderungen,

sondern auch zum Beispiel durch Refactoring oder Optimierung auftreten. Dies führt zu einer weiteren Verminderung der Transparenz. In dieser Arbeit soll aufgezeigt werden, wie durch Verwendung natürlichsprachlicher Ausdrücke Änderungen in einer Datenbank wieder transparent gemacht werden können. Eine grundlegende Anforderung an das zu realisierende System war, dass auch für Nichtexperten die Änderungen in Datenbeständen verständlich sind, und in einer „Historie“ für jeden Datensatz angezeigt werden können. Hierfür werden durchgeführte Datenbanktransaktionen aufgezeichnet und in grammatisch und im Kontext korrekte, verbale Ausdrücke übersetzt. Ein Beispiel eines solchen generierten natürlichsprachlichen Ausdrucks ist der folgende:

Die Rufnummer des Telefonbucheintrags mit dem Namen „Herr Müller“ wurde von 0451 654321 in 0451 123456 geändert.

Ein entsprechendes System zur Aufzeichnung von Datenbanktransaktionen und anschliessenden Übersetzung in natürlichsprachliche Ausdrücke wurde im Rahmen einer Diplomarbeit realisiert, auf welcher diese Arbeit basiert [Bu02]. Die Diplomarbeit am Institut für Informationssysteme der Universität zu Lübeck wurde bei der Fa. Pietsch - Organisationsberatung und Informationstechnologie - durchgeführt. Bei der Fa. Pietsch handelt es sich um ein Lübecker Unternehmen, tätig in der Beratung und der Systementwicklung für die Bereiche des Umweltschutzes, der Arbeitssicherheit, des vorbeugenden Brandschutzes sowie des Liegenschaftsmanagements. Das Unternehmen versteht sich aufgrund der konsequenten Spezialisierung im Gesundheitswesen als Dienstleister für das Krankenhaus. Um optimierte Lösungen krankenhausspezifischer Aufgabenstellungen vornehmlich für den Bereich der Aufbau- und Ablauforganisation verwirklichen zu können, wurden spezielle Managementsysteme durch die Fa. Pietsch projektiert und entwickelt, die sich unter anderem auch in mehreren Universitätskliniken im Einsatz befinden. Das hier entwickelte System wurde nach Durchführung der Diplomarbeit weiterentwickelt, und ist seitdem ebenfalls in mehreren Universitätskliniken deutschlandweit im Einsatz.

Eine weitere wesentliche Anforderung an das System war, dass die Performanz bestehender Softwarekomponenten nicht wesentlich beeinträchtigt werden durfte. Zur Darstellung eines Verlaufs von Feldwerten wurde in einem Dialog den Benutzern das Verfolgen der generierten lesbaren Texte ermöglicht.

2 Aufzeichnung von Transaktionen

Bevor eine Übersetzung von Datenbanktransaktionen in natürliche Sprache vorgenommen werden kann, müssen diese in einem dafür geeigneten Format aufgezeichnet werden. Alternativ könnte dies auch direkt nach Durchführung der Transaktion geschehen, indem die entsprechenden Daten an eine Systemkomponente übergeben werden. Dies ist jedoch aufgrund der Anforderungen an die Performance und die Flexibilität sowie Wartbarkeit des Systems wenig vorteilhaft.

Änderungen in einer Datenbank geschehen stets durch eine einer sequentiellen Abfolge äquivalenten Abarbeitung von schreibenden Transaktionen. Diese werden üblicherweise in einem sogenannten Logfile protokolliert. Konzeptuell ist dies in der Regel eine herkömmliche Tabelle, an die sequentiell entsprechend der zeitlichen Abfolge Einträge

angehängt werden. Dazu notwendige Algorithmen und Systeme zum Management werden in der Literatur, wie zum Beispiel [GR93] oder [HR01], beschrieben. Dabei sind die Inhalte der Einträge im Logfile abhängig von den Anforderungen an die Transaktionsaufzeichnung. Auf die Aufzeichnung lesender Operationen kann für die Verbalisierung verzichtet werden, da diese keine Änderungen in der Datenbank verursachen. Für schreibende Transaktionen werden hier stets sämtliche durchgeführten Änderungen an der Datenbank aufgezeichnet, wobei zwischen den drei grundlegenden Zurgriffsoperationen APPEND (Neuanlage von Datensätzen/Feldern), CHANGE (Verändern von Datensätzen/Feldern) und DELETE (Löschen von Datensätzen) unterschieden wird. Ein Logfile, mit dem diese Anforderungen für relationale Datenbanken realisiert werden können, sollte aus einer Abfolge von folgenden Tupeln bestehen: Transaktion (Eindeutiger Schlüssel für Zusammengehörigkeit von Operationen), Ursprung (Zeigt Urheberschaft der Operation auf, z.B. Zeitpunkt und Benutzer), Tabelle, Datensatz, Spalte, Alter Wert und Neuer Wert. Jedes Tupel identifiziert hierbei eine einzelne, elementare Zugriffsoperation, die zu einer Feldänderung geführt hat. Jedes Feld einer relationalen Datenbank kann hierbei durch seine Position (Spalte,Datensatz,Tabelle) eindeutig identifiziert werden. Zur Darstellung werden die elementaren Zugriffsoperationen READ(field) und WRITE(field,value) verwendet, welche den Wert eines Feldes lesen bzw. schreiben (vgl. hierzu auch [Vos00]).

3 Generierung natürlichsprachlicher Übersetzungen

In der Computerlinguistik ist ein grundlegendes systematisches Vorgehen die Unterscheidung zwischen Inhaltsbestimmung (deciding what to say) und einer anschließenden Formbestimmung (deciding how to say it). Diese Vorgehensweise wurde auch hier gewählt.

Eine wesentliche Aufgabe bei der Inhaltsbestimmung zur Sprachgenerierung ist die Wortwahl. Dazu muss für ein Konzept der internen Repräsentation ein passendes Lexem ausgewählt werden. Als Lexem wird hier eine lexikalische Bedeutungseinheit bezeichnet, die eine fassbare Reflexion eines zugehörigen Gedankens bezeichnet. Diese Begrifflichkeit ist bei verschiedenen Autoren jedoch teilweise anders belegt[1], in jedem Fall haben Lexeme jedoch eine eindeutige sprachliche Identität. Zur Bestimmung des Inhaltes wurde für relationale Datenbanken das E/R-Modell nach Peter Chen gemäß [BCN91] zugrundegelegt. Sowohl in der Modellierungsphase, als auch in der schematischen Abbildung auf ein DBMS werden Bezeichner zur eindeutigen Identifizierung von Elementen wie Attributen, Entitäten oder Tabellen verwendet. Ein konkretes Beispiel wäre der Primärschlüssel einer Tabelle. Diese Bezeichner sollten einen klaren Ausdruck über den Inhalt des Elementes vermitteln, sind allerdings nicht notwendigerweise von der Zielsprache der Verbalisierung belegt. In der Praxis werden oft Abkürzungen oder aber auch Worte anderer Sprachen verwendet, z.B. Anglizismen wie User für den deutschen Begriff Benutzer. Das folgende Beispiel veranschaulicht dieses:

[1] Vielfach werden Lexeme auch als Spezialisierungen von Wortarten betrachtet.

Der eindeutige Bezeichner PERS_TBL, der eine Tabelle bezeichnet, liefert eine gewisse Information über die Funktion dieses Elementes des Datenbankschemas. Er ist aber sprachlich nicht eindeutig belegt, und somit ist PERS_TBL für eine Übersetzung ungeeignet.

Wie im Abschnitt 2 beschrieben, kann man Felder in relationalen Datenbanken durch Tabellen, Spalten und Zeilen eindeutig identifizieren. Für diese drei Elemente sollen nun eindeutige Lexeme gefunden werden. Diese „Namen" sollen weiterhin nicht notwendigerweise mit den Bezeichnern des Schemas oder der Modellierung übereinstimmen müssen. Durch die Verwendung von Lexemen ist gesichert, dass diese auch für eine Übersetzung weiterverwendet werden können, da jedes eine eindeutige sprachliche Identität besitzt. Lexeme für Tabellen sollen hier als L_x definiert werden. Ein Lexem L_x für eine Spalte ist analog dem Bezeichner für die Tabelle definiert. Ein Feld kann eindeutig identifiziert werden, wenn die zugehörige Tabelle T_z, Zeile R_y und Spalte C_x bekannt sind. Sind eindeutige Bezeichner hierfür bekannt, so kann die Position eines Feldes auch durch diese eindeutig bestimmt werden, z.B.:

Die Rufnummer des Telefonbucheintrages mit dem Namen „Herr Müller" ist 12345.

Die sequentielle Abfolge von Datensätzen (Zeilen) in einer Tabelle wird gewöhnlich durch eine Sortierung mit Indizes bestimmt. Ein Index wird über eine oder mehrere Spalten (Attribute) einer Tabelle definiert und enthält die entsprechenden Feldwerte. Verfügt eine Tabelle über einen eindeutigen, dichten[2] Index, so kann jeder Datensatz anhand des Indexeintrages eindeutig identifiziert werden [Vos00]. Solch ein eindeutiger, dichter Index kann zum Beispiel der Primärschlüssel einer Relation sein. Das Finden eines Bezeichners für Zeilen in Form eines eindeutigen Lexems ist aber aufgrund der Variabilität der Indexeinträge nicht möglich. Ein eindeutiger Bezeichner kann aber für die zugrundeliegenden Spalten analog zu den existierenden Lexemen für die Einzelspalten gefunden werden. Dieser eindeutige Bezeichner L_y wird mit dem Indexeintrag kombiniert, welcher sich aus den zugehörigen Feldwerten zusammensetzt. Eindeutige Bezeichner können ebenso in objektorientierten Datenbanken gefunden werden.

Ein weiterer Teil der Inhaltsbestimmung ist neben der Ermittlung lexikalischer Inhalte die Identifizierung von pragmatischen Zusammenhängen. Zur Bestimmung von Beziehungen und Inhalten in einer Datenbank sollen daher klar definierte Kontexte verwendet werden. Ein Kontext beschreibt einen Zusammenhang zwischen verschiedenen Feldern oder Änderungen innerhalb einer Datenbank. Ein Kontext kann abhängig von sämtlichen Parametern sein, die sich in einer Aufzeichnung wie z.B. einem Logfile finden lassen. Der zugehörige Teil des Übersetzungssystems muss also in der Lage sein, Kontexte anhand der Einträge des Logfiles zu identifizieren. Dies kann durch den Vergleich mit vorher abgelegten Mustern für jeden Kontext (bestehend z.B. aus Feld, Tabelle, Wert, ...) geschehen. Für eine praktische Umsetzung würde sich bei einer großen Anzahl von Kontexten eine Indizierung der Datenstruktur anbieten, um diese effektiv auffinden zu können.

---

[2] In einem dichten (dense) Index findet sich im Gegensatz zu einem dünnen (sparse) Index für jeden Wert der betreffenden Attribute ein Eintrag.

Datenbanktransaktionen setzen sich aus mehreren Einzeloperationen zusammen (vgl. Abschnitt 2). Somit kann eine Transaktion durch Übersetzung der zugehörigen Einzeloperationen in natürliche Sprache übertragen werden.

Durch Generierung von natürlichsprachlichen Übersetzungen für jeden dadurch entstehenden neuen Kontext kann dann eine Historie der zugehörigen Datensätze respektive Feldwerte aufgebaut werden. Die so generierten Beschreibungen können in einer eigenen Tabelle oder anderen Form mit den zugehörigen Bezügen abgelegt werden. Von dort sind diese dann bei Bedarf durch einfache Anfragen oder Einschränkungen entsprechend der zugehörigen Elemente des Datenbankschemas abrufbar.

Neben der Inhaltsbestimmung ist eine weitere Aufgabe von Sprachgenerierung die Erzeugung von Kennzeichnungen zur Repräsentation von sprachlichen Elementen. Hier soll die deutsche Sprache verwendet werden, da das realisierte System ebenfalls deutschsprachig ist. Die im folgenden vorgestellten Verfahren können aber auch für alle Sprachen verwendet werden, die über eine feste Form und Grammatik verfügen.

Durch die Auswahl von exemplarischen Beispielen für den jeweiligen Kontext kann eine Satzstruktur durch eine Syntaxanalyse abgeleitet werden. Die syntaktische Struktur der Beispielsätze wird dann in eine Abfolge von Symbolen zerlegt, welche jeweils einzelne lexikalische Elemente repräsentieren. Diese können dann bei der eigentlichen Generierung durch Ersetzungsregeln durch die zugehörigen natürlichsprachlichen Ausdrücke ersetzt werden. Aus diesen Symbolen lassen sich dann für jeden Kontext Schablonen zur Generierung von grammatisch korrekten Sätzen erzeugen. Kontexte in Datenbanken können zum Beispiel Änderung, Neueinträge oder Löschungen sein. Zur Beschreibung eines Kontextes werden neben den zugehörigen Bezeichnern für die Position des Feldes, auf das sich die Beschreibung bezieht, auch Werte für dieses Feld benötigt. Eine Übersetzung wird daher auf Grundlage mehrerer Kontexte gebildet. Jedem dieser Kontexte ist eine Schablone zugeordnet, die sich aus einer Sequenz von Symbolen zusammensetzt, welche passende Übersetzungsregeln repräsentieren.

Schablonen, feststehende Lexeme und Ersetzungsregeln eines Generators zur Erzeugung von Übersetzungen können in einer geeigneten Datenstruktur, wie einer Tabelle, gespeichert werden. Wortbildungsfunktionen wie zum Beispiel die Deklinationsfunktion oder die Bildung von Artikeln können auch, je nach Sprache, durch Verwendung von hierfür geeigneten Algorithmen gebildet werden. Das Auffinden solcher Formalismen ist in den Bereichen der Linguistik bzw. Computerlinguistik Gegenstand aktueller Forschungen. Die Entscheidung, ob Wortbildungsfunktionen in einer Datenstruktur abgelegt werden, die dann entsprechend interpretiert wird, oder durch geeignete einfache Funktionen bzw. Algorithmen in ein Übersetzungssystem fest integriert werden, sollte je nach Umfang und Komplexität der zu generierenden Wortformen getroffen werden. Dies ist vor allem auch abhängig von der Differenziertheit der Wortbildungsmöglichkeiten der Zielsprache. Verfügt eine Sprache zum Beispiel über sehr viele unregelmäßig gebildete Wortformen, so ist die zusätzliche Verwaltung eines entsprechenden Wörterbuches neben einem Regelwerk unerlässlich.

Die Schaffung eines komplexen Generatorsystems für grammatische Formen ist für eine geringe Anzahl von Kontexten und verwendeten Lexemen sicher nicht gerechtfertigt. Stattdessen können Lexeme in einem einheitlichen Wörterbuch verwaltet werden. Beim Hinzufügen neuer Einträge sollten grammatische Formen, wie zum Beispiel die Deklination, durch grundlegende Regeln basierend auf elementaren Eigenschaften wie Genus oder Numerus gebildet und anschließend durch Benutzer des Systems überprüft und eventuell korrigiert werden. Die Verwendung von Schablonen stellt eine einfach zu realisierende Lösung dar, die ebenfalls zu einer sehr guten Performance des Gesamtsystems führt. Einzelne Elemente wie Symbolfolgen werden hierbei, ähnlich einzelner Ansätzen von Expertensystemen, jedoch mehrfach verwendet oder gespeichert. Durch die Verwendung von Formalismen, wie zum Beispiel Produktionsregeln, ließe sich dies vermeiden. Ein anderer Ansatz als die direkte Verwendung von vorher abgelegten Schablonen wäre also der Aufbau von Schablonen mit Hilfe einer regulären Grammatik, wobei noch zu überprüfen wäre, inwieweit dies auch bei Expertensystemen Verwendung finden kann. Hierzu soll dann anstatt der gesamten Schablone eine entsprechende Startproduktion entsprechend des Kontextes gewählt werden. Die Produktionsregeln von Grammatiken erzeugen dann die fest determinierte Satzstruktur, welche die Positionen einzelner Elemente strikt festlegt. Terminalsymbole entsprechen den Symbolen der Schablonen, wie sie bereits festgelegt wurden. Die zugehörigen Ersetzungsregeln füllen diese Symbole dann mit lexikalischem Material. Nonterminalsymbole können zum Teil den syntaktischen Elementen des zu generierenden Satzes entsprechen.

Für die Verbalisierung „gewöhnlicher“ Datenbanktransaktionen, auch mit einigen Sonderfällen, ist die direkte Verwendung von Schablonen ausreichend. Diese können dann zum Beispiel parallel zu den Mustern zur Kontexterkennung abgelegt werden. Da innerhalb größerer Anwendungssysteme eine große Zahl von Transaktionen durchgeführt wird, liegt in der Verwendung von Schablonen nicht nur ein Vorteil für die Performance des Gesamtsystems, sondern es resultiert auch eine bessere Wartbarkeit und Wiederverwendbarkeit. Bei der Verwendung von Grammatikformalismen zum Aufbau von Schablonen sollte dann stattdessen eine Speicherung der zugehörigen Startsymbole und der entsprechenden Produktionen erfolgen.

Ein Kriterium, das die allgemeine Verständlichkeit der generierten Sätze wesentlich beeinflusst, sind die Inhalte der Feldwerte. Bestehen diese aus Zeichenketten, wie zum Beispiel Eigennamen, so ist dies unkritisch zu betrachten. Auch Zahlenwerte wie zum Beispiel Währungsbeträge können in einen sinnvollen Zusammenhang gebracht werden. Repräsentiert ein Feldwert jedoch einen Schlüssel aus einer anderen Tabelle so ist eine für den Menschen allgemeinverständliche Übersetzung nicht direkt möglich.

Der Titel der Person mit dem Namen „Herr Müller“ wurde von 1 in 2 geändert.

In Datenbanken können verschiedene Tabellen durch Einführen von Fremdschlüsseln miteinander verknüpft werden. Diese werden durch einen eindeutigen, dichten Index geordnet und können auch den Primärschlüssel repräsentieren. Diese Werte können dann in ein entsprechendes Feld des gleichen Typs einer anderen Tabelle übernommen werden, um entsprechende Verknüpfungen abzubilden. Werden in den

Datenbanktabellen Fremdschlüssel verwendet, so können diese durch den entsprechenden Primärschlüssel der zugehörigen Verknüpfungstabelle ersetzt werden:

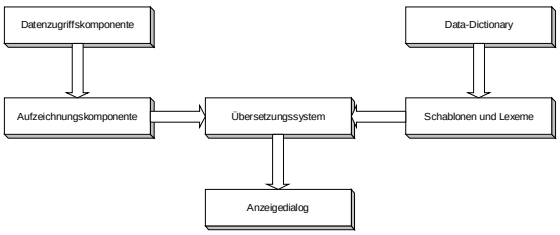
Die Rufnummer des Telefonbucheintrages mit der Person 4 wurde von 4 in 3 geändert.

Dieser generierte Text ist zwar für den zugehörigen Kontext korrekt, verfügt aber über eine geringe Verständlichkeit. Ersetzt man nun sämtliche Einträge von Feldwerten, die Fremdschlüssel von anderen Tabellen repräsentieren, durch den zugehörigen Datensatzbezeichner der Fremdtabelle, so wird die Verständlichkeit wesentlich erhöht:

Die Rufnummer des Telefonbucheintrages mit dem Namen „Herr Meier“ wurde von 0451 987654 in 0451 456789 geändert.

4 Implementierung und Performance-Aussagen

Die Implementierung des Systems erfolgte objektorientiert unter Verwendung der Borland Delphi IDE unter Microsoft® Windows NT / 2000. Dieses setzt sich aus drei wesentlichen Komponenten zusammen: Die Erweiterung eines bestehenden Data-Dictionaries zur Verwaltung von Lexemen und Patterns, eine Komponente zur Protokollierung abgeschlossener Datenbanktransaktionen, eine Serverapplikation[3] zur Generierung der Übersetzungen und ein Anzeigedialog für die übersetzten Einträge



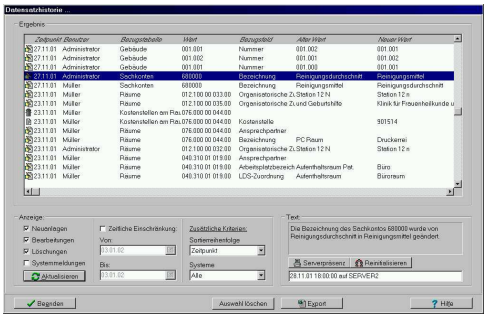
Bei der Erweiterung des bestehenden Data-Dictionaries handelte es sich im wesentlichen um die Bereitstellung von Eingabemasken für die benötigten Daten. Hierzu zählen ein Wörterbuch für die Lexeme, die Verknüpfung dieser Lexeme an bestehende Metadaten wie Tabellen oder Spalten sowie eine Möglichkeit zur Verwaltung von Schablonen zur Übersetzungsgenerierung.

Das System verwendet eine clientbasierte Protokollierungskomponente, die unter der Verwendung des Listener-Musters (vgl. hierzu [GHJV96]) auf bestehende Datenbankzugriffskomponenten aufsetzt, ohne diese zu verändern. Erfolgreich durchgeführte Transaktionen, und damit die zugehörigen Feldänderungen, werden in eine sequentiellen Tabelle als Logfile gespeichert. In einer ersten, testweisen Implementierung wurde festgestellt, dass eine Protokollierungskomponente ohne jegliche Zwischenspeicherung erhebliche Performanceverluste zur Folge hatte. Dies lag darin begründet, dass jede einzelne aufgezeichnete Operation mehrere schreibende Operationen für die Eintragung in das Logfile nach sich zog und auf mehreren Clients zu fast identischen Zeitpunkten auftrat. Dadurch wurde das DBMS durch Spitzen in der Anzahl der Zugriffe stark belastet. Dies verlangsamte nicht nur das Antwortverhalten der entsprechenden Einzelapplikation merklich, sondern beeinflusste auch das Antwortverhalten von auf derselben Datenbank operierenden Anwendungen negativ.

[3] Eine Serverapplikation ist eine Anwendung, die jeweils nur einmal auf einem Applikationsserver ausgeführt wird. Diese kann entweder ein direkt lauffähiges Programm sein, oder z.B. ein NT-Dienst oder Unix-Daemon.

Dieses entsprach nicht den Anforderungen. Daher musste eine Möglichkeit zur Zwischenspeicherung des Logfiles vor Ablage in die Datenbank geschaffen werden, welche Leerlaufzeiten der Einzelapplikationen und des DBMS ausnutzt. Hierzu bietet das Konzept des Threads[4] (Leichtgewichtsprozess, vgl. hierzu [So98]) unter Windows NT eine mögliche Lösung. Threads mit der niedrigsten Prioritätsstufe erhalten fast ausschließlich nur bei „Leerlauf“ des Betriebssystems Rechenzeit zugeteilt. Die Protokollierungskomponente wurde daher zweigeteilt: Die eigentliche Aufzeichnungskomponente bleibt als Hauptobjekt in die Applikation eingebunden, während der für die Logfileeintragungen zuständige Teil als diesem zugeordneter Thread gestaltet wird. Beide verwenden einen gemeinsamen Speicherbereich, in dem sich eine Liste mit den zu protokollierenden Änderungen befindet. Diese wird von der Protokollierungskomponente bei Auslösen der entsprechenden Ereignisse gefüllt, während ein Thread diese in das Logfile einträgt. Da dies aufgrund der niedrigen Priorität nur in Leerlaufzeiten der zugehörigen Applikation geschieht, konnte der Performanceverlust so wesentlich verringert werden. Allerdings werden zur Synchronisation weitere Mechanismen benötigt, um Verklemmungen durch gleichzeitigen Zugriff auf die gemeinsame Liste zu verhindern. Hierzu wurde eine Semaphor-ähnliche Lösung zum wechselseitigen Ausschluss verwendet. Die Gesamtsystemkomponente zeigte dann ein akzeptables Laufzeitverhalten.

Zur Verbalisierung werden von der Serverapplikation die im Data-Dictionary definierten Schablonen und Lexeme verwendet. Die verbalisierten Texte werden dann in einer weiteren, separaten Tabelle abgelegt. Die eigentliche Applikation wird durch zwei vertikal angeordnete Klassen aufgebaut. Die obere der Klassen ist ein Formular mit einer Verlaufsansicht zur Überwachung der Applikation. Sie ist ferner für eine zyklische Abfrage der Logfile-Tabelle, die Kontextidentifikation und das Schreiben der übersetzen Einträge zuständig. Die zweite Klasse übernimmt das Übersetzen der Einträge. Übersetzt werden die Kontexte APPEND (Hinzufügen von Datensätzen), CHANGE (Ändern von Feldwerten) und DELETE (Löschen von Datensätzen). Für jeden Kontext existiert eine Schablone, die in entsprechenden Methoden fest codiert ist. Dazu wird ein im Data-Dictionary definiertes Pattern gesucht, welches Ersetzungsregeln für die einzelnen Symbole der Schablonen liefert. Zur Sicherstellung der Revisionssicherheit werden an jeder Übersetzung zusätzlich Benutzer, Uhrzeit und Teilsystem vermerkt, die ursächlich für die Datenänderung waren.



Die mit der Serverapplikation generierten Übersetzungen sollen für Benutzer angezeigt werden. Hierzu wird für diesen Teil der Benutzerschnittstelle ein Standarddialog verwendet, der in einem Anzeigegitter die übersetzten Texte anzeigt. Diese werden durch ein SQL-Statement abgefragt, in dessen „WHERE-Bereich“ sich vom Benutzer festgelegte Bedingungen finden. Angezeigt werden

[4] Analoge Konzepte existieren auch unter anderen Betriebssystemen, wie z.B. Unix.

neben dem generierten Text auch die Bezeichner (Lexeme) von an der Übersetzung beteiligten Feldwerten, sowie der Zeitpunkt. Ferner werden Einschränkungs- und Anordnungsmöglichkeiten geschaffen, die eine bessere Übersichtlichkeit verschaffen sollen. Zum Aufruf dieses Dialoges existieren zwei Methoden: Aufruf ohne Parameter (sämtliche generierten Texte werden abgefragt), sowie Aufruf mit den Parametern „Tabellenname“, „Datensatznummer“ (nur die Einträge der Tabelle und des zugehörigen Datensatzes werden abgefragt). Letzteres stellt die eigentliche *Datensatzhistorie* dar.

5 Vergleich mit anderen Arbeiten

Auf die Aufzeichnung von Datenbanktransaktionen in Logfiles wird in der Standardliteratur, wie [Vos00], [GR93] oder [HR01] ausführlich eingegangen. In [GR93] werden verschiedene Modelle und Mechanismen zum Management von Transaktionsaufzeichnungen angeführt und erläutert.

Zum Aufbau der Produktionsregeln und Symbolmengen, und somit der automatischen Zergliederung eines Satzes, können Algorithmen und Strategien aus dem Bereich der Computerlinguistik verwendet werden. Solche Systeme werden u.a. auch als Parser bezeichnet. [Co01] gibt einen Überblick über bestehende Lösungsansätze sowie aktuelle Forschungen.

Weitere Arbeiten zum Thema Sprachgenerierung sind [RD95] sowie [RD00], welche einen Überblick über Generierung natürlicher Sprache aus Sicht der praktischen Systementwicklung gibt. Auch im Bereich der KI[5] werden schon seit 25 Jahren Systeme zur Generierung natürlicher Sprache entwickelt. Insbesondere gilt dies für Erklärungskomponenten von Expertensystemen [BS84]. Diese sind in der Regel sehr mächtig und daher aus Effizienzgründen für die Verbalisierung von Datenbanktransaktion wenig geeignet, da hierbei für sehr viele Transaktionen sehr rasch Übersetzungen generiert werden müssen. [MB98] enthält unter anderem detaillierte Beiträge zu den Themen der Generierung und Verarbeitung von Sprache. In der „*Bibliography of Research in Natural Language Generation*“ findet sich eine umfassende Sammlung von Referenzen zur Generierung natürlicher Sprache: http://liinwww.ira.uka.de/bibliography/Ai/nlg.html.

6 Ausblick

Nicht immer werden in einem System sämtliche Daten in einer Datenbank gespeichert, auch wenn dies vor allem dem Ziel einer weitgehenden Datenunabhängigkeit dienlich wäre. Oftmals ist dies der Fall bei rechnerspezifischen Einstellungen (z.B. der Bildschirmauflösung). Zur Erweiterung des Systems könnten auch solche Änderungen durch Verwendung einer weiteren Methode aufgezeichnet werden. Dies erfordert allerdings stets einen gewissen Implementierungsaufwand, so dass dies nur für ausgewählte Änderungen getan werden sollte.

[5] Künstliche Intelligenz

Die durch die Serverapplikation generierten Texte können anhand von weiteren Parametern der Schablonen gefiltert werden. Bestimmte so selektierte Übersetzungen können dann in Form von Nachrichten an einzelne Benutzer oder Benutzergruppen versendet werden. Hierzu können bereits bestehende Nachrichtendienste verwendet werden. Mit Hilfe dieses Mechanismus können einzelne Benutzer, z.B. Administratoren, bei einzelnen Datenänderungen gezielt informiert werden. Nützlich könnte dies u. A. auch zur Erhöhung der Sicherheit des Gesamtsystems sein. Auch revisionsrelevante Daten, wie Kosten bestimmter Dienstleistungen können so durch weitere Personen direkt nachvollzogen und eventuell erneut nachgeprüft werden.

Literaturverzeichnis

[ABHS99] Antos G.; Brinker, K.; Heidemann, W.; Sager, S.F.: Text und Gesprächslinguistik, 2. Band "Gesprächslinguistik" de Gruyter Berlin 1999

[BCN91] Batini, C.; Cesi, S.; Navathe, S.B.: Conceptual Database Design: An Entity-Relationship Approach. Benjamin/Cummings Pub. Co. 1991

[BS84] Buchanan, B.; Shortliffe, E. (Eds): Rule-Based Reasoning: the MYCIN Experiment . Addison-Wesley, Reading, MA, 1984

[Ch00] Christodoulakis, Dimitris (Ed.): Natural Language Processing - NLP 2000. Second Int. Conference, Patras, Greece, June 2-4, 2000, Proceedings. Lecture Notes in Computer Science 1835 Springer 2000

[BDDS99] Brandt, P.; Dettmer, D.; Dietrich, R.A.; Schön, G.: Sprachwissenschaft. Böhlau Verlag 1999

[Bu02] Buck, Olaf: Verbalisierung von Datenbanktransaktionen. Diplomarbeit am Institut für Informationssysteme der Universität zu Lübeck 2002, erschienen als Techn. Bericht B-02-05 der Institute für Informatik und Mathematik der Universität zu Lübeck 2002

[Co91] Cole; R.A.: Survey of the State of the Art in Human Language Technology. http://cslu.cse.ogi.edu/HLTsurvey

[GHJV96] Gamma; Helm; Johnson; Vlissides: Entwurfsmuster. Addison Wesley 1996

[GR93] Gray, J.; Reuter A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers 1993

[Hau92] Hausser, R.: Complexity in Left-Associative Grammar Theoretical Computer Science 106(2), 283-308. Dortrecht: Elsevier

[Hoe99] Hoeppner, W.: Der Mensch-Maschine Dialog http://www.uni-duisburg.de/FB3/CL/seitenD/Veroeffentlichungen/MM-Dialog.html und in [ABHS99]

[HR01] Härder, T.; Rahm E.: Datenbanksysteme – Konzepte und Techniken der Implementierung. Springer Verlag 2001

[MB98] McDonald, D.D.; Bolc L. (Ed.): Natural Language Generation Systems. Springer-Verlag 1998

[RD95] Reiter, Ehud; Dale, Robert: Building Applied Natural Language Generation Systems. Cambridge University Press 1995

[RD00] Reiter, Ehud; Dale, Robert: Building Natural Language Generation Systems. Cambridge University Press 2000

[So98] Solomon, David: Inside Windows NT. Microsoft Press 1998

[Vos00] Vossen, Gottfried: Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme. München, Oldenbourg Verlag

Integritätsbedingungen für komplexe Objekte in objektrelationalen Datenbanksystemen

Jens Lufter
Friedrich-Schiller-Universität Jena
lufter@informatik.uni-jena.de

Abstract: Ein Aspekt objektrelationaler Datenbanksysteme ist die Möglichkeit zur Definition von strukturierten und von Kollektionsdatentypen und die sich daraus ergebende Verwendbarkeit komplex geschachtelter Objekte. SQL-Norm und wichtige DBMS-Produkte haben jedoch noch deutliche Defizite in der Sicherung der Integrität komplex aufgebauter Attribute von Datenbanktabellen. Der vorliegende Beitrag diskutiert Anforderungen an eine adäquate Integritätssicherung, beschreibt den aktuellen Stand der Dinge in SQL-Norm und wichtigen Produkten und untersucht Möglichkeiten zur Ergänzung von SQL um Sprachmittel für entsprechende Integritätsbedingungen.

1 Einführung

Viele Anwendungssysteme halten ihre Daten heute in relationalen Datenbanksystemen, der Zugriff darauf erfolgt über die normierte Datenbanksprache SQL [ISO99] bzw. produktspezifische Dialekte derselben. Der Einfluss der Objektorientierung hat mittlerweile zur evolutionären Weiterentwicklung von relationalen zu objektrelationalen Datenbanksystemen geführt [DD98, Luf99], deren Grundlage erweiterbare Datentypsysteme sind.

Ein wichtiger Aspekt ist dabei die Schachtelbarkeit von Datentypen, insbesondere von strukturierten und von Kollektionsdatentypen. Damit können potentiell sehr tief geschachtelte Konstrukte entstehen. Noch nicht gut gelöst sind in diesem Zusammenhang u. a. Probleme der Integritätssicherung. In relationalen Datenbanksystemen ist diese Frage auf der Ebene von Datenbanktabellen und deren Attributen angesiedelt, spezialisierte Integritätsbedingungen (`NOT NULL`, Schlüssel, Fremdschlüssel) referenzieren diese Attribute, können jedoch nicht in ihren möglicherweise komplexen Aufbau „hineinschauen".

SQL-Norm und wichtige Produkte bieten heute die Möglichkeit, strukturierte und Kollektionsdatentypen zu definieren und orthogonal zueinander zu verwenden. Die Definition von Kollektionsdatentypen ist jedoch meist unzureichend und sehr heterogen gelöst und bedarf deutlicher Verbesserungen [Luf03b]. Wie die folgenden Ausführungen zeigen, ist es außerdem erforderlich, einen hinreichenden Umfang von Sprachmitteln zur Integritätssicherung anzubieten, die nicht nur auf Attributebene wirken, sondern tiefer in die Struktur komplex geschachtelter Objekte eindringen können. Das vorgestellte Thema ist Teil des Projekts *Object-Relational Database Features and Extensions: Data Model and Physical Aspects*, das durch einen *IBM Shared University Research Grant* unterstützt wird.

2 Integritätsbedingungen für komplexe Objekte

2.1 Datentypen in objektrelationalen Datenbanksystemen

Objektrelationale Datenbanksysteme erlauben die Definition von nutzerdefinierten Datentypen und Routinen und die Konstruktion von Datentypen mit Hilfe von Typgeneratoren. Zu den nutzerdefinierten Typen zählen dabei *Distinct types* zur strengen Typisierung anderer Datentypen und die strukturierten nutzerdefinierten Datentypen, die objektorientierte Eigenschaften wie Methoden, Objektidentität und -referenzen sowie Vererbung und Polymorphie aufweisen. Letztere können neben der üblichen Verwendung von Datentypen, z. B. zur Attributdefinition von Datenbanktabellen, auch zur Spezifikation typisierter Tabellen genutzt werden, deren Tupel Objekte dieses Typs inklusive Objektidentität und Referenzierbarkeit darstellen.

Zu den generierten (konstruierten) Datentypen zählen Kollektions-, Struktur- und Referenztypen (`ARRAY, SET, ROW, REF` usw.). Typgeneratoren ermöglichen die Definition von Datentypen, die ein gemeinsames allgemeines Verhalten, also vergleichbare Operationen, aufweisen. Parameter eines Typgenerators sind Datentypen, Feldnamen oder Längenangaben, das spezifische Verhalten des entstehenden Typs wird diesen Parametern automatisch angepasst. Generierte Datentypen sind im Gegensatz zu nutzerdefinierten Datentypen nicht Elemente des Datenbankschemas.

Nutzerdefinierte und generierte Typen können orthogonal zur Definition weiterer Datentypen und zur Attributdefinition von Datenbanktabellen genutzt werden. Dadurch können potentiell tief geschachtelte Typstrukturen entstehen, entsprechende Typinstanzen bezeichnen wir hier auch als komplexe Objekte. Um im Rahmen von Datenbankzugriffen auf innere Strukturen komplexer Objekte zugreifen zu können, müssen adäquate Operationen bereitstehen. So kann man Attribute von Strukturtypen mittels Punkt-Notation ansprechen, für Kollektionen gibt es Operationen zum Elementzugriff oder das Umwandeln in eine Tabelle (`UNNEST`), auf die dann relationale Operationen anwendbar sind.

2.2 Komplexe Attribute in Datenbanktabellen

Zur Illustration der Probleme bei der Integritätssicherung für komplexe Attribute von Datenbanktabellen soll die folgende Tabellendefinition dienen. Das Beispiel verwendet vordefinierte Datentypen wie `INTEGER`, generierte Strukturtypen (`ROW`), eine Reihe generierter Kollektionstypen (`ARRAY, SET, LIST`) und Referenzdatentypen zum Verweis auf Objekte in typisierten Tabellen (`REF`). Der Übersicht wegen abstrahieren wir von nutzerdefinierten Datentypen; im Hinblick auf die Integritätssicherung sind *Distinct types* zum jeweils zugrundeliegenden Datentyp äquivalent, nutzerdefinierte strukturierte Typen zu `ROW`. Die Kollektionsunterstützung in SQL-Norm und Produkten ist derzeit noch nicht befriedigend und sehr heterogen umgesetzt. Die Ausführungen beziehen sich daher auf ein von uns entwickeltes erweitertes Kollektionskonzept [Luf03b], dessen genauere Beschreibung hier jedoch zu weit führen würde.

```
CREATE TABLE Angestellter (
  ID         IdType PRIMARY KEY,
  Name       ROW ( Nachname CHAR(20),
                   Vornamen LIST ( CHAR(20) ) ),
  Adressen   ROW ( PLZ CHAR(5), Ort CHAR(20) ) ARRAY[3],
  Telefone   SET ( CHAR(20) ),
  MailListen SET ( LIST ( IdType ) ),
  Kinder     SET ( REF ( KindTyp ) ) )
```

Angestellte haben somit zunächst eine identifizierende Nummer und einen aus Nachname und Vornamen zusammengesetzten Namen. Ein Angestellter kann ggf. mehrere Vornamen haben, deren Ordnung wichtig ist, eine Liste bietet sich an. Weiterhin seien bis zu drei Adressen in einem Feld von zweiattributigen Strukturen verzeichnet. Außerdem möge ein Angestellter eine Menge von Telefonnummern haben. Geschachtelte Kollektionen illustriert die Menge der Mail-Listen eines Angestellten, jede Mail-Liste ist wiederum eine geordnete Kollektion von Angestellten-IDs. Die Kinder eines Angestellten ergeben sich aus einer Menge von Referenzen auf Objekte einer typisierten Tabelle. Bis auf den Primärschlüssel enthält die Tabelle noch keine Integritätsbedingungen.

2.3 Klassifikation von Integritätsbedingungen

In relationalen Datenbanksystemen gibt es traditionell drei wesentliche Klassen von Integritätsbedingungen: Schlüssel, Fremdschlüssel und allgemeine, über Anfragen definierte `CHECK`-Bedingungen. Hinzu kommt auf Grund der dreiwertigen Logik von SQL die Kurzschreibweise für `NOT NULL`-Bedingungen. Zur Unterscheidung von allgemeinen Bedingungen wollen wir Konstrukte wie Schlüssel, Fremdschlüssel oder `NOT NULL` im Folgenden auch als spezialisierte Integritätsbedingungen bezeichnen.

Schlüssel- und Fremdschlüsselbedingungen können auf allgemeine Bedingungen zurückgeführt werden, in der Tat macht das die SQL-Norm bei der Beschreibung der Semantik dieser Bedingungen genau so [ISO99]. Dennoch gibt es für diese beiden Fälle eine eigene Notation, die sich zum einen natürlich aus dem relationalen Datenmodell selbst ergibt. Zum zweiten ist die Notation deutlich bequemer zu handhaben und bietet einfachere Möglichkeiten für eine effiziente Implementierung. Schließlich gibt es noch semantische Zusätze. So kann eine Tabelle nur einen Primärschlüssel haben und die daran beteiligten Attribute können nicht `NULL` werden, ein Schlüssel ist zudem als Ziel von Fremdschlüsseln verwendbar. Wichtig sind die referentiellen Aktionen für Fremdschlüssel (`ON DELETE CASCADE` usw.), die aktives Verhalten in die Integritätssicherung einführen.

Objektrelationale Datenbanken ergänzen die genannten Sprachmittel um ein weiteres zum Absichern von Objektreferenzen. In unserem Beispiel ist `REF ( KindTyp )` ein Datentyp, dessen Werte auf Tupel von mit `KindTyp` typisierten Tabellen zeigen können, potentiell kann es mehrere solcher Tabellen geben. In Anlehnung an Fremdschlüssel kann man das Referenzziel auf eine (oder auch mehrere) Tabellen beschränken (`SCOPE`) und zudem, ebenfalls analog zu Fremdschlüsseln, referentielle Aktionen ergänzen.

Auf Grund der traditionell armen Typsysteme bisheriger relationaler Datenbanksysteme setzen Schlüssel und Fremdschlüssel, aber auch NOT NULL-Bedingungen, bislang auf der Attributebene von Tabellen an, können also nicht in komplexere Strukturen eindringen. Allgemeine Bedingungen können dagegen die ganze Anfragemächtigkeit von SQL ausnutzen. Bevor wir die Notwendigkeit spezialisierter Konstrukte für die Integritätssicherung unter Einbeziehung komplexer Objekte diskutieren, daher zwei Beispiele für allgemeine Bedingungen, die an sich einfache Aufgaben simulieren. Die erste der Regeln zeigt, wie Fremdschlüsselbedingungen simuliert werden können: Die Einträge der Mail-Listen gehören zu vorhandenen Angestellten. Die zweite Regel stellt sicher, dass die Menge der Telefone kein NULL-Element enthalten darf.

```
ALTER TABLE Angestellter
  ADD CONSTRAINT MailingIDs CHECK ( NOT EXISTS
    ( SELECT *
      FROM   Angestellter a,
             UNNEST ( a.MailListen ) AS ml ( Liste ),
             UNNEST ( ml.Liste ) AS l ( MailID )
      WHERE  l.MailID NOT IN
               ( SELECT ID FROM Angestellter ) )

ALTER TABLE Angestellter
  ADD CONSTRAINT TelefonNotNull CHECK ( NOT EXISTS
    ( SELECT *
      FROM   Angestellter a,
             UNNEST ( a.Telefone ) AS t ( Telefon ),
      WHERE  t.Telefon IS NULL )
```

UNNEST ist dabei immer die SQL-Operation, die eine Kollektion in eine relationale Tabelle verwandelt, der Rest der Beispiele sollte selbsterklärend sein. Schon der Formulierungsaufwand zeigt, dass für wichtige Regelarten spezialisierte Konstrukte eingeführt werden sollten. Zudem kann die Fremdschlüsselsimulation des ersten Beispiels nicht um referentielle Aktionen ergänzt werden.

Für komplexe Objekte können neue Arten von Integritätsbedingungen erforderlich sein [The96], wesentlich sind im objektrelationalen Fall aber vor allem Erweiterungen für Schlüssel-, Fremdschlüssel- und NOT NULL-Bedingungen, so dass sie in geschachtelte Datentypen eindringen können. Wichtige Beispiele für Integritätsbedingungen unter Einbeziehung geschachtelte Strukturen sind:

- Elemente einer Kollektion dürfen nicht NULL werden

  Es ist natürlich ein Unterschied, ob ein kollektionswertiger Ausdruck NULL werden kann, oder ob einzelne Elemente einer Kollektion NULL sind.

- Felder einer Struktur dürfen nicht NULL werden

  Mit ROW (...) IS NOT NULL kann man in SQL zwar schon ganze Strukturen ansprechen, dieses Konstrukt wird aber so umgesetzt, dass dann alle Felder der Struktur nicht NULL werden dürfen. Sinnvoll wäre auch ein selektives Verhalten.

- Schlüsselbedingungen für Datenbanktabellen

  Schlüsselbedingungen können eine vielfältige Gestalt haben. So kann der Schlüssel einer Datenbanktabelle z. B. auch Felder einer Struktur enthalten, etwa den Nachnamen in unserem Beispiel, oder auch einzelne Elemente einer Kollektion wie den ersten Vornamen aus der Vornamensliste eines Angestellten.

- Fremdschlüsselbedingungen

  Eine einfache Fremdschlüsselbedingung kann sich z. B. auf ein mengenwertiges Tabellenattribut beziehen und fordern, dass die Elemente der Menge sich auf den Schlüssel einer anderen Tabelle beziehen: `...SET ( INTEGER REFERENCES AndereTabelle )...` Ähnliches gilt für Felder von Strukturen innerhalb eines komplexen Attributs. Fremdschlüssel können referentielle Aktionen umfassen.

- Gesicherte Objektreferenzen

  Das Ziel von Objektreferenzen soll auf ausgewählte typisierte Tabellen einschränkbar sein, analog zu Fremdschlüsseln sollen referentielle Aktionen erlaubt sein.

- Duplikatverbot und andere Eindeutigkeitsbedingungen

  Für ungeordnete Kollektionen ohne Duplikate gibt es Mengentypen, Duplikatfreiheit kann aber auch für geordnete Kollektionen wichtig sein. Beispiele für Eindeutigkeitsbedingungen in (ggf. tiefer geschachtelten) Kollektionen sind etwa Schlüssel für Attributkombinationen geschachtelter Tabellen.

3 SQL-Norm und Produkte

Die objektrelationalen Möglichkeiten zur Nutzung komplexer Objekte in SQL-Norm und wichtigen Produkten unterscheiden sich bislang noch erheblich. Neben einer vernünftigen Unterstützung von Kollektionen mangelt es vor allem noch an der Integritätssicherung für geschachtelte Strukturen.

An für unsere Untersuchungen relevanten objektrelationalen Sprachmitteln erlaubt SQL in der aktuellen Version von 1999 [ISO99] die Nutzung generierter und nutzerdefinierter Strukturtypen (`ROW`, *Structured type*) und von generierten, allerdings nicht schachtelbaren, Feld-Typen (`ARRAY`). Außerdem gibt es Objektreferenzen zum Referenzieren von Tupeln typisierter Tabellen (`REF`). Die Folgenorm SQL:200x [ISO02] wird Multimengen ergänzen und Kollektionen schachtelbar machen.

Mit SQL:1999 kann man in Schlüsseln, Fremdschlüsseln und `NOT NULL`-Bedingungen nur ganze Attribute einer Datenbanktabelle referenzieren, innere Teile komplex strukturierter Attribute sind nicht ansprechbar. Man kann sie allenfalls so wie in den Beispielen aus Abschnitt 2.3 mit `CHECK`-Bedingungen simulieren.

Objektreferenzen lassen sich auf eine Zieltabelle beschränken, außerdem sind referentielle Aktionen angebbar. Dies gilt nicht nur für Referenzen auf Attributebene, sondern auch für eingeschachtelte Referenzen in Feldern von Strukturtypen (`ROW`, nutzerdefinierte struktu-

rierte Typen). Als Elementtyp von Kollektionen lässt sich Referenztypen zwar die Zieltabelle mitgeben, es sind aber keine referentiellen Aktionen angebbar:

```
CREATE TABLE myTable (
  att1 REF ( type1 ) SCOPE table1
         REFERENCES ARE CHECKED ON DELETE CASCADE,
  att2 LIST ( ROW ( field1 REF ( type2 ) SCOPE table2
                      REFERENCES ARE CHECKED
                        ON DELETE NO ACTION ) ),
  att3 SET ( REF ( type3 ) SCOPE table3 ) )
```

Die Semantik-Beschreibung der Norm wandelt dabei Referenz-Bedingungen auf oberster Ebene (`att1`) in Fremdschlüssel um, Bedingungen eingeschachtelter Referenzen in `CHECK`-Bedingungen. Das impliziert auch, dass im zweiten Fall nur `NO ACTION` angebbar ist, da referentielle Aktionen nur für Fremdschlüssel definiert sind.

Das nicht orthogonale Verhalten ergibt sich aus einem Design-Fehler der Norm [LFP02]. Insgesamt ist festzustellen, dass SQL bislang keine adäquaten Konzepte zur Integritätssicherung komplexer Objekte umfasst, das wird sich nach dem derzeitigen Stand der Dinge auch mit der Folgenorm nicht ändern [ISO02].

Wichtige aktuelle Produkte wie Oracle 9i, Informix Dynamic Server 9.3 oder DB2 UDB 7.3 erlauben in sehr unterschiedlichem Umfang die Nutzung von Struktur- und Kollektionsdatentypen zur Definition komplexer Datenstrukturen. Möglichkeiten zur Definition adäquater Integritätsbedingungen für komplexe Objekte gibt es so gut wie nicht. Aus Platzgründen kann hier kein entsprechender Überblick gegeben werden, der interessierte Leser sei auf [Luf03a] verwiesen.

4 Wege zur Ergänzung der SQL-Norm

Im folgenden stellen wir einen Ansatz zur Integration spezialisierter Integritätsbedingungen für komplexe Objekte in die SQL-Norm vor. Dabei liegt der Schwerpunkt auf syntaktischen Überlegungen, semantische Fragen und Details zur Einbindung in das Normdokument können hier nicht näher erläutert werden.

4.1 Zugriff auf die innere Struktur geschachtelter Datentypen

Für die gewünschten Integritätsbedingungen muss man die innere Struktur geschachtelter Datentypen ansprechen können, das geschieht durch eine Erweiterung der Punkt-Notation, mit der man in SQL auf Felder von Strukturtypen zugreifen kann. Im Gegensatz zu Feldern von Strukturtypen sind die Elementtypen von Kollektionsdatentypen unbenannt. Um sie ansprechen zu können, nutzen wir das Schlüsselwort `ELEMENT`, die Punkt-Notation erlaubt dann die Angabe von Pfaden zum Eindringen in eine komplexe Struktur.

Um also den Elementtyp der Vornamensliste unseres Angestellten-Beispiels aus Abschnitt 2.2 in einer Integritätsbedingung ansprechen zu können, muss man über das Vornamensfeld der Namensstruktur gehen, analog kann man auf Strukturen in Kollektionstypen (Ort einer Adresse) oder geschachtelte Kollektionen (ID in einer Mail-Liste) zugreifen:

```
Name.Vornamen.ELEMENT
Adressen.ELEMENT.Ort
MailListen.ELEMENT.ELEMENT
```

Abgedeckt ist natürlich auch das Ansprechen ganzer Kollektionen oder Strukturen, etwa der Vornamensliste oder der Adressstruktur:

```
Name.Vornamen
Adressen.ELEMENT
```

Semantisch muss man sich die Nutzung von `ELEMENT` wie ein implizites `UNNEST` vorstellen. Grundlage von Integritätsbedingungen sind ja letztlich Anfragen, auch die SQL-Norm spezifiziert die Semantik spezialisierter Integritätsbedingungen durch deren Umwandlung in allgemeine, anfragebasierte `CHECK`-Bedingungen. Wenn man also verlangen will, dass die Vornamen nicht `NULL` werden dürfen (`Name.Vornamen.ELEMENT NOT NULL`), führt das zu einer äquivalenten allgemeinen Bedingung:

```
CHECK ( NOT EXISTS
  ( SELECT *
    FROM   Angestellter a,
           UNNEST ( a.Name.Vornamen ) AS v ( Vorname ),
    WHERE  v.Vorname IS NULL ) )
```

Neben der neuen `ELEMENT`-Syntax, die eine Kollektion sozusagen „flachklopft“ und alle ihre Elemente (einzeln) anspricht, erlaubt SQL:1999 bereits den Zugriff auf einzelne Elemente von Felddatentypen innerhalb von Pfadausdrücken, unsere erweiterte Kollektionsunterstützung [Luf03b] erlaubt das auch für Listen. Damit ist es möglich, Integritätsbedingungen an spezifische Elemente einer Liste zu knüpfen statt an alle, etwa an den ersten Vornamen eines Angestellten oder den Ort seiner zweiten Adresse:

```
Name.Vornamen[1]
Adressen[2].Ort
```

Man kann somit mit einer einfachen und klaren Syntax Teile komplexer Datenstrukturen ansprechen, die Änderungen in den Semantik-Teilen der Norm lassen sich aber nichtsdestotrotz in Grenzen halten.

4.2 Erzeugen und Löschen von Integritätsbedingungen

Die im vorigen Abschnitt eingeführte Syntax macht es auf einfache Weise möglich, Bedingungen für komplexe Strukturen auf Attribut- und Tabellenebene zu definieren bzw.

bei Tabellenänderungen zu löschen: Auf Attributebene würde man z. B. bei der Definition des Namensattributs mit `Vornamen.ELEMENT` den Elementtyp der Vornamensliste referenzieren, auf Tabellenebene schriebe man `Name.Vornamen.ELEMENT`.

Ein Aspekt soll hier noch angesprochen werden, nämlich das Aufheben der Vermischung von intensionalen und extensionalen Fragen, wie sie in Abschnitt 3 kurz diskutiert wurden: Für Objektreferenzen werden in der SQL-Norm Zieltabelle und dazugehörige Integritätsbedingungen teilweise intensional, also auf Datentypebene definiert. Das führt neben Problemen mit der Orthogonalität auch zu Inkonsistenzen bei Änderungen: Man kann mit `ALTER TABLE` schlecht Datentypen verändern [LFP02]. Solche Bedingungen werden mit unserem Ansatz ebenfalls auf die Attribut- bzw. Tabellenebene verlagert.

4.3 Spezifizierbare Integritätsbedingungen

Das von uns entwickelte Konzept erlaubt derzeit die Einbeziehung von komplex strukturierten Attributen in vier Arten spezialisierter Integritätsbedingungen: `NOT NULL`-Bedingungen, Schlüssel, Fremdschlüssel und gesicherte Objektreferenzen. Das Mischen von einfach und komplex strukturierten Attributen ist dabei (wo sinnvoll) möglich.

Die genaue Syntaxspezifikation und Fragen der exakten Umsetzung in der Norm würden den Umfang dieses Beitrags sprengen und sind teilweise auch noch Gegenstand laufender Arbeiten. Statt dessen sollen Anwendung und Mächtigkeit anhand des aus Abschnitt 2.2 bekannten Angestellten-Beispiels erläutert werden:

```
CREATE TABLE Angestellter (
  ID          IdType PRIMARY KEY,
  Name        ROW ( Nachname CHAR(20),
                    Vornamen LIST ( CHAR(20) ) )
                CONSTRAINT c1 NOT NULL
                CONSTRAINT c2 Vornamen.ELEMENT NOT NULL,
  Adressen    ROW ( PLZ CHAR(5), Ort CHAR(20) ) ARRAY[3]
                CONSTRAINT c3 ELEMENT.Ort NOT NULL,
  Telefone    SET ( CHAR(20) ),
  MailListen SET ( LIST ( IdType ) )
                CONSTRAINT c4 ELEMENT NOT NULL
                CONSTRAINT c5
                  ELEMENT.ELEMENT REFERENCES Angestellter,
  Kinder      SET ( REF ( KindTyp ) )
                CONSTRAINT c6
                  ELEMENT SCOPE KindTabelle
                          REFERENCES ARE CHECKED
                            ON DELETE CASCADE,
  CONSTRAINT c7
    UNIQUE ( Name.Nachname, Name.Vornamen[1],
             Adressen[1] ) )
```

Es sei noch einmal darauf hingewiesen, dass das Beispiel neben den neuen Integritätsbedingungen auch Bezug auf eine erweiterte Kollektionsunterstützung nimmt [Luf03b]. Die Praxisrelevanz der spezifizierten Integritätsbedingungen sei Nebensache.

Zunächst zu den `NOT NULL`-Bedingungen. Bedingung c1 ist eine herkömmliche SQL-Bedingung, sie gilt für das ganze Attribut. Da dieses strukturiert ist, impliziert die SQL-Semantik, dass weder der Nachname noch die Vornamensliste `NULL` sein dürfen. Was noch fehlt, ist eine Bedingung, dass auch die einzelnen Vornamen dieser Liste bekannt sein müssen (c2). Bedingung c3 verlangt, dass die Orte von Adressen spezifiziert sein müssen. Im Gegensatz dazu können sowohl einzelne Postleitzahlen als auch einzelne Adressen und ebenso das ganze Adressen-Attribut `NULL` werden. Schließlich besagt c4, dass die Menge der Mail-Listen keine undefinierten Elemente enthalten darf.

Einen einfachen Fremdschlüssel spezifiziert Bedingung c5: Die Elemente einer Mail-Liste referenzieren wiederum Angestellte. Bedingung c6 sichert schließlich Objektreferenzen ab. Im Vergleich zu SQL:1999 gibt es hier drei Neuheiten: Die Norm kann zum einen direkt in Kollektionen enthaltene Referenzen bislang gar nicht absichern, das ist jetzt möglich. Zum zweiten sind für eingeschachtelte Referenzen nun auch beliebige referentielle Aktionen spezifizierbar. Und zum dritten sind die Angabe der Zieltabelle und die Bedingung jetzt Teil der Tabellendefinition, nicht mehr der Typdefinition.

Bedingung c7 ist ein Beispiel für einen zusätzlichen Schlüssel, der zudem die Verwendung von Element-Referenzen demonstriert, wie sie SQL:1999 bereits eingeführt hat. Die Bedingung besagt, dass eine Kombination aus Nachname, erstem Vornamen und erster Adresse eines Angestellten eindeutig sein soll.

4.4 Integration in die Norm

Unsere Arbeiten zur genaueren Spezifikation der erforderlichen Normerweiterungen orientieren sich bislang an SQL:1999, werden aber anhand der jeweils aktuellen Arbeitsversionen zur Folgenorm SQL:200x verifiziert. Neben der vergleichsweise einfachen syntaktischen Umsetzung liegt dabei der Schwerpunkt auf der Einbindung der Semantik in die bestehende Infrastruktur für Integritätsbedingungen in der Norm.

Die hier vorgestellten Konzepte sollen zudem anhand einer derzeit prototypisch entwickelten Transformationsschicht validiert werden, die erweiterte objektrelationale Sprachmittel auf reale Datenbanksysteme abbildet [KLS02, Her03].

5 Zusammenfassung und Ausblick

Objektrelationale Datenbanksysteme bieten heute die Möglichkeit zur Definition komplex geschachtelter Objekte mit Hilfe von strukturierten und von Kollektionsdatentypen, wenn auch auf recht heterogene Art und Weise. Potentiell tief geschachtelte Datenstrukturen erfordern aber auch neue Möglichkeiten der Integritätssicherung, hier sind noch deutliche

Verbesserungen in SQL-Norm und Produkten nötig. Der vorliegende Beitrag hat versucht, auf Basis von Anforderungen an solche Integritätsbedingungen und den gegenwärtigen Stand der Technik Wege zur Ergänzung der aktuellen SQL-Norm vorzustellen, die diesen Erfordernissen genügen.

Ende 2003 wird voraussichtlich die nächste Version der SQL-Norm verabschiedet. Der aktuelle Arbeitsstand (*Final Committee Draft*, [ISO02]) macht eine Aufnahme der Vorschläge bereits in diese Version der Norm unwahrscheinlich, das ändert aber nichts an deren Validität und Notwendigkeit. Neben entsprechenden Zuarbeiten für das Normungsgremium [FLP02, LFP02] sowie einer detaillierteren Spezifikation und ggf. Ergänzung der vorgestellten Norm-Erweiterungen ist es das Ziel weiterer Arbeiten, die vorgestellten Sprachmittel in Verbindung mit einer erweiterten Unterstützung für Kollektionen prototypisch umzusetzen und zu validieren.

Literatur

[DD98] C. J. Date and H. Darwen. *Foundation for Object/Relational Databases: The Third Manifesto*. Addison-Wesley, Reading, MA, 1998.

[FLP02] T. Fanghänel, J. Lufter, and P. Pistor. ISO/IEC JTC 1/SC32/WG3 DRS-091: Introducing Some Indicators Missing from Definition Schema, August 2002.

[Her03] A. Hermann. Prototypische Umsetzung erweiterter objektrelationaler Datenbankkonzepte durch Abbildung auf reale DBMS. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2003. In Vorbereitung.

[ISO99] ISO/IEC 9075:1999. *Information Technology – Database Languages – SQL*, 1999.

[ISO02] ISO/IEC Final Committee Draft 9075-2:200x. *Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation)*, January 2002.

[KLS02] C. Kauhaus, J. Lufter, and S. Skatulla. Eine Transformationsschicht zur Realisierung objektrelationaler Datenbankkonzepte mit erweiterter Kollektionsunterstützung. *Datenbank-Spektrum*, 2(4):49–58, November 2002.

[LFP02] J. Lufter, T. Fanghänel, and P. Pistor. ISO/IEC JTC 1/SC32/WG3 DRS-089: Problems related to <scope clause> and <reference scope check>, September 2002.

[Luf99] J. Lufter. Objektrelationale Datenbanksysteme (Aktuelles Schlagwort). *Informatik Spektrum*, 22(4):288–290, August 1999.

[Luf03a] J. Lufter. Integritätsbedingungen für komplexe Objekte in objektrelationalen Datenbanksystemen. Jenaer Schriften zur Mathematik und Informatik, Institut für Informatik, Friedrich-Schiller-Universität Jena, January 2003.

[Luf03b] J. Lufter. Kollektionsunterstützung für SQL:1999. *Informatik - Forschung und Entwicklung*, 2003. Angenommen zur Veröffentlichung.

[The96] S. Thelemann. Semantische Anreicherung eines Datenmodells für komplexe Objekte. Dissertation, FB 17 Mathematik/Informatik, Universität Gesamthochschule Kassel, June 1996.

The Paradigm of Relational Indexing: A Survey

Hans-Peter Kriegel[1], Martin Pfeifle [1], Marco Pötke[2] and Thomas Seidl[3]
[1]University of Munich, {kriegel, pfeifle}@dbs.informatik.uni-muenchen.de
[2]sd&m AG software design & management, marco.poetke@sdm.de
[3]Aachen University (RWTH), seidl@informatik.rwth-aachen.de

Abstract: In order to achieve efficient execution plans for queries comprising user-defined data types and predicates, the database system has to be provided with appropriate index structures, query processing methods, and optimization rules. Although available extensible indexing frameworks provide a gateway to seamlessly integrate user-defined access methods into the standard process of query optimization and execution, they do not facilitate the actual implementation of the access method itself. An internal enhancement of the database kernel is usually not an option for database developers. The embedding of a custom block-oriented index structure into concurrency control, recovery services and buffer management would cause extensive implementation efforts and maintenance cost, at the risk of weakening the reliability of the entire system. The server stability can be preserved by delegating index operations to an external process, but this approach induces severe performance bottlenecks due to context switches and inter-process communication. Therefore, we present in this paper the paradigm of relational access methods that perfectly fits to the common relational data model and is highly compatible with the extensible indexing frameworks of existing object-relational database systems.

1 Introduction

The design of extensible architectures represents an important area in database research. The object-relational data model marked an evolutionary milestone by introducing abstract data types into relational database servers. Thereby, object-relational database systems may be used as a natural basis to design an integrated user-defined database solution. The ORDBMSs already support major aspects of the declarative embedding of user-defined data types and predicates. In order to achieve a seamless integration of custom object types and predicates within the declarative DDL and DML, ORDBMSs provide the database developer with extensibility interfaces. They enable the declarative embedding of abstract data types within the built-in optimizer and query processor. Corresponding frameworks are available for most object-relational database systems, including Oracle [Ora 99a] [SMS+00], IBM DB2 [IBM99] [CCF+99], or Informix IDS/UDO [Inf98] [BSSJ99]. Custom server components using these built-in services are called *data cartridges*, *database extenders*, and *data blades*, in Oracle, DB2 and Informix, respectively.

In this paper, we categorize possible approaches to incorporate third-party indexing structures into a relational database system what we call *Relational Indexing*. Following this introduction about ORDBMS and their extensible indexing facilities, Section 2 discusses three different implementations of user-defined access methods, including the relational approach. In Section 3, basic concepts of relational access methods are introduced, and in Section 4, the design of the corresponding update and query operations are investi-

```
// Type declaration

CREATE TYPE POINT AS OBJECT (x NUMBER, y NUMBER);
CREATE TYPE POINT_TABLE AS TABLE OF POINT;
CREATE TYPE POLYGON AS OBJECT (
  points POINT_TABLE,
  MEMBER FUNCTION intersects (p POLYGON) RETURN BOOLEAN
);

// Type implementation
// …

// Functional predicate binding

CREATE OPERATOR INTERSECTS (a POLYGON, b POLYGON)
RETURN BOOLEAN
BEGIN RETURN a.intersects(b); END;

// Table definition

CREATE TABLE polygons (id NUMBER PRIMARY KEY, geom POLYGON);
```

Figure 1: Object-relational DDL statements for polygon data

gated. In Section 5, we identify two generic schemes for modeling relational access methods which are discussed with respect to their support of concurrent transactions and recovery. The paper is concluded in Section 6.

Declarative Integration. As an example, we create an object type *POLYGON* to encapsulate the data and semantics of two-dimensional polygons. Instances of this custom object type are stored as elements of relational tuples. Figure 1 depicts some of the required object-relational DDL statements in pseudo SQL thus abstracting from technical details which depend on the chosen product. By using the functional binding of the user-defined predicate *INTERSECTS*, object-relational queries can be expressed in the usual declarative fashion (cf. Figure 2). Provided only with a functional implementation which evaluates the INTERSECTS predicate in a row by row manner, the built-in optimizer has to include a full-table scan into the execution plan to perform the spatial selection. In consequence, the resulting performance will be very poor for highly selective query regions. As a solution, the extensibility services of the ORDBMS offer a conceptual framework to supplement the functional evaluation of user-defined predicates with index-based lookups.

```
// Region query

SELECT id FROM polygons
WHERE INTERSECTS(geom, :query_region);
```

Figure 2: Object-relational region query on polygon data for a region *query_region*

Function	Task
index_create(), index_drop()	Create and drop a custom index.
index_open(), index_close()	Open and close a custom index.
index_fetch()	Fetch the next record from the index that meets the query predicate.
index_insert(), index_delete(), index_update()	Add, delete, and update a record of the index.

Figure 3: Methods for extensible index definition and manipulation

Extensible Indexing. An important requirement for applications is the availability of user-defined access methods. Extensible indexing frameworks proposed by Stonebraker [Sto86] enable developers to register custom secondary access methods at the database server in addition to the built-in index structures. An object-relational *indextype* encapsulates stored functions for creating and dropping a custom index and for opening and closing index scans. The row-based processing of selections and update operations follows the iterator pattern [GHJV95]. Thereby, the indextype complements the functional implementation of user-defined predicates. Figure 3 shows some basic indextype methods invoked by extensible indexing frameworks. Additional functions exist to support query optimization, custom joins, and user-defined aggregates. Assuming that we have encapsulated a spatial access method for two-dimensional polygons within the custom indextype *SpatialIndex*, we may create an index *polygons_idx* on the *geom* attribute of the *polygons* table by submitting the usual DDL statement (cf. Figure 4). If the optimizer decides to include this custom index into the execution plan for a declarative DML statement, the appropriate indextype functions are called by the built-in query processor of the database server. Thereby, the maintenance and access of a custom index structure is completely hidden from the user, and the desired data independence is achieved. Furthermore, the framework guarantees any redundant index data to remain consistent with the user data.

Talking to the Optimizer. Query optimization is the process of choosing the most efficient way to execute a declarative DML statement. Object-relational database systems typically support rule-based and cost-based query optimization. The extensible indexing framework comprises interfaces to tell the built-in optimizer about the characteristics of a custom indextype. Figure 5 shows some cost-based functions, which can be implemented to provide the optimizer with feedback on the expected index behavior. The computation

```
// Index creation

CREATE INDEX polygons_idx ON polygons(geom)
INDEXTYPE IS SpatialIndex;
```

Figure 4: Creation of a custom index on polygon data

Function	Task
stats_collect(), stats_delete()	Collect and delete persistent statistics on the custom index.
predicate_sel()	Estimate the selectivity of a user-defined predicate by using the persistent statistics.
index_cpu_cost(), index_io_cost()	Estimate the CPU and I/O cost required to evaluate a user-defined predicate on the custom index.

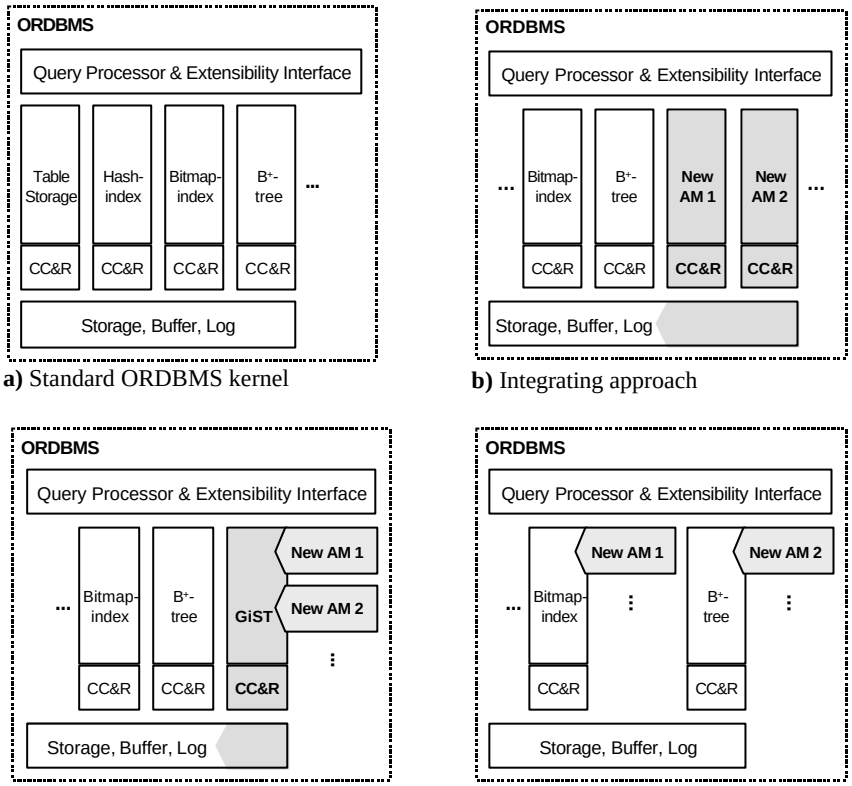
Figure 5: Methods for extensible query optimization

of custom statistics is triggered by the usual administrative SQL statements. With a cost model registered at the built-in optimizer framework, the cost-based optimizer is able to rank the potential usage of a custom access method among alternative access paths. Thus, the system supports the generation of efficient execution plans for queries comprising user-defined predicates. This approach preserves the declarative paradigm of SQL, as it requires no manual query rewriting.

2 Implementation of Access Methods

In the previous section, we have outlined how object-relational database systems support the logical embedding of custom indextypes into the declarative query language and into the optimizer framework. The required high-level interfaces can be found in any commercial ORDBMS and are continuously improved and extended by the database vendors. Whereas the embedding of a custom indextype is therefore well supported, its actual implementation within a fully-fledged database kernel remains an open problem. In the following, we discuss three basic approaches to implement the low-level functionality of a user-defined access method: the *integrating*, the *generic*, and the *relational approach* (cf. Figure 6).

Integrating Approach. By following the integrating approach, a new access method (AM) is hard-wired into the kernel of an existing database system (cf. Figure 6b). In consequence, the required support of ACID properties, including concurrency control and recovery services (CC&R) has to be implemented from scratch and linked to the corresponding built-in components. Furthermore, a custom gateway to the built-in storage, buffer, and log managers has to be provided by the developer of the new AM. Most standard primary and secondary storage structures are hard-wired within the database kernel, including plain table storage, hash indexes, bitmap indexes, and B+-trees. Only a few non-standard access methods have been implemented into commercial systems in the same way, including the *R-Link-tree* in Informix IDS/UDO for spatially extended objects [Inf99] and the *UB-tree* in TransBase/HC for multidimensional point databases [RMF+00]. The integrating approach comprises the *Extending Approach* and the *Enhancing Approach.* The extending approach is expensive, since it is really adding a new access method plus all the concurrency, locking, etc. (R-Link-Tree, Bitmaps, External Memory Interval Tree). In contrast to this the enhancing approach is much cheaper, since most properties get inherited, e.g. enhancing B-Trees to be functional B-Trees. We identify the following properties of the integrating approach:



a) Standard ORDBMS kernel

b) Integrating approach

c) Generic approach

d) Relational approach

Figure 6: Approaches to implement custom access methods

Implementation: The implementation of a new AM becomes very sophisticated and tedious if writing transactions have to be supported [Bro01]. In addition, the code maintenance is a very complex task, as new kernel functionality has to be implemented for any built-in access method. Moreover, the tight integration within the existing kernel source produces a highly platform-dependent solution tailor-made for a specific ORDBMS.

Performance: The integrating approach potentially delivers the maximal possible performance, if the access method is implemented in a closed environment, and the number of context switches to other components of the database kernel is minimized.

Availability: The implementation requires low-level access to most kernel components. If the target ORDBMS is not distributed as open-source, the affected code and documentation will not be accessible to external database developers.

To sum up, the integrating approach is the method of choice only for a few, well-selected access methods serving the requirements of general database applications. It is not feasible for the implementation of too specialized access methods.

Generic Approach. To overcome the restrictions of the integrating method, Hellerstein, Naughton and Pfeffer [HNP95] proposed a generic approach to implement new access methods in an ORDBMS. Their *Generalized Search Tree* (*GiST*) has to be built only once

into an existing database kernel. The GiST serves as a high-level framework to plug in block-based tree structures with full ACID support (cf. Figure 6c). Many extensions to the GiST framework have been presented, including generic support for concurrency and recovery [KMH97], and additional interfaces for nearest-neighbor search, ranking, aggregation, and selectivity estimation [Aok98]. In detail, the GiST approach has the following characteristics:

Implementation: Whereas the implementation of block-based access methods on top of the GiST framework can be done rather easily, the intruding integration of the framework itself remains a very complex task. As an advantage, an access method developed for GiST can basically be employed on any ORDBMS that supports this framework. In contrast to the generic GiST implementation, the specialized functionality of a new access method is therefore platform independent.

Performance: Although the framework induces some overhead, we can still achieve a high performance for GiST-based access methods. Kornacker [Kor99] has shown that they may even outperform built-in index structures by minimizing calls to user-defined functions.

Availability: Due to its complex implementation, the GiST framework is only available as a research prototype. It is an open question, if and when a comparable functionality will be a standard component of major commercial ORDBMSs.

The GiST concept basically delivers the desired properties to implement custom access methods. It delegates crucial parts of the implementation to the database vendors. To our best knowledge, however, its full functionality is not available on any major database system. Furthermore, database extensions should generically support many database platforms. Thus, the GiST concept would have to be implemented not only for one, but for all major ORDBMS.

Relational Approach. A natural way to avoid the above obstacles is to map the custom index structure to a relational schema organized by built-in access methods (cf. Figure 6d). Such *relational access methods* are designed to operate on top of a relational query language. They require no extension or modification of the database kernel, and, thus, any off-the-shelf ORDBMS can be employed as it is. We identify the following advantages for the relational approach:

Implementation: As no internal modification or extension to the database server is required, a relational access method can be implemented and maintained with less effort. Substantial parts of the custom access semantics may be expressed by using the declarative DML. Thereby, the implementation exploits the existing functionality of the underlying ORDBMS rather than duplicating basic database services as done in the integrating and generic approaches. Moreover, if we use a standardized DDL and DML like SQL:1999 [SQL99] to implement the low-level interface of our access method, the resulting code will be platform independent.

Performance: The major challenge in designing a relational access method is to achieve both a high usability and performance. In [KPS00] [KPS01] [KMPS01a] [KMPS01b] the capability and efficiency of the relational approach was proven for interval data and 2D/3D spatial data.

Availability: By design, a relational access method is supported by any relational database system. It requires the same functionality as an ordinary database user or a relational database application.

By following the relational approach to implement new access methods, we obtain a natural distinction between the basic services of all-purpose database systems and specialized, application-specific extensions. By restricting database accesses to the common SQL interface, custom access methods and query procedures are well-defined on top of the core server components. In addition, a relational access method immediately benefits from any improvement of the ORDBMS infrastructure.

3 Basics of Relational Access Methods

The basic idea of relational access methods relies on the exploitation of the built-in functionality of existing database systems. Rather than extending any internal component of the database kernel, a relational access method just uses the native data definition and data manipulation language to process updates and queries on abstract data types. Without loss of generality, we assume that the underlying database system implements the standardized *Structured Query Language* SQL-92 [SQL92] with common object-relational enhancements in the sense of SQL:1999 [SQL99], including object types and collections.

3.1 Paradigms of Access Methods

A relational access method delegates the management of persistent data to an underlying relational database system by strictly implementing the index definition and manipulation on top of an SQL interface. Thereby, the SQL layer of the ORDBMS is employed as a *virtual machine* managing persistent data. Its robust and powerful abstraction from block-based secondary storage to the object-relational model can then be fully exploited. This concept also perfectly supports database appliances, i.e. dedicated database machines running the ORDBMS as a specialized operating system [KP92] [Ora00]. We add the class of relational access methods as a third paradigm to the known paradigms of access methods for database management systems:

Main Memory Access Methods (Figure 7a). Typical applications of these techniques can be found in main memory databases [DKO+85] [GS92] and in the field of computational geometry [PS93]. A popular example taken from the latter is the binary *Interval Tree* [Ede80]. It serves as a basic data structure for plane-sweep algorithms, e.g. to process intersection joins on rectangle sets. Main memory structures are not qualified for indexing persistent data, as they disregard the block-oriented access to secondary storage.

Block Oriented Access Methods (Figure 7b). These structures are designed to efficiently support the block-oriented I/O from and to external storage and are well suited to manage large amounts of persistent data. The *External Memory Interval Tree* [AV96] is an example for the optimal externalization of a main memory access method. Its analytic optimality is achieved by adapting the fanout of the Interval Tree to the disk block size. In the absence of a generalized search tree framework [HNP95], the implementation of such specialized storage structures into existing database systems, along with custom concurrency control and recovery services, is very complex, and furthermore, requires intrusive modifications of the database kernel [RMF+00].

Relational Access Methods (Figure 7c). In contrast, relational access methods including the *Relational Interval Tree* [KPS00] are designed to operate on relations rather than on

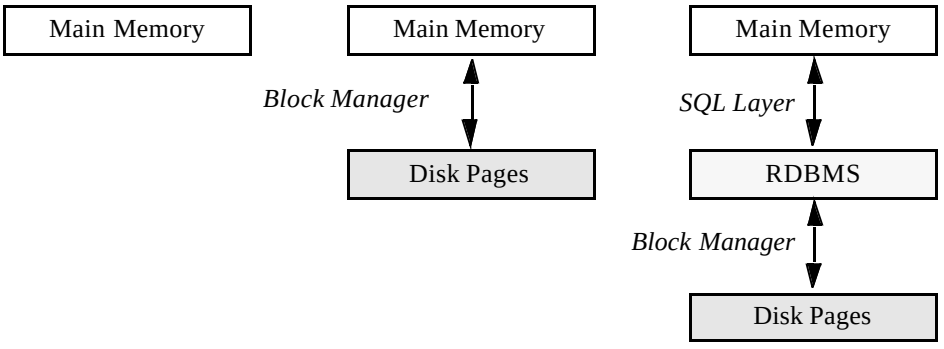


Figure 7: Paradigms and characteristics of access methods: **a)** main memory access methods, **b)** block-oriented access methods, and **c)** relational access methods.

dedicated disk blocks. The persistent storage and block-oriented management of the relations is delegated to the underlying database server. Therefore, the robust functionality of the database kernel including concurrent transactions and recovery can potentially be reused. A primary clustering index can be achieved by also delegating the clustering to the ORDBMS. For this, the payload data has to be included into the index relations and the clustering has to be enabled by organizing these tables in a cluster or as index-organized tables [SDF+ 00].

3.2 Relational Storage of Index Data

In the remainder of this paper, we will discuss the basic properties of relational access methods with respect to the storage of index data, query processing and the overhead for transaction semantics, concurrency control, and recovery services. We start with a basic definition:

Definition 1 (*Relational Access Method*).
An access method is called a *relational access method*, iff any index-related data is exclusively stored in and retrieved from relational tables. An instance of a relational access method is called a *relational index*. The following tables comprise the persistent data of a relational index:

(*i*) *User table*: a single table, storing the original user data being indexed.
(*ii*) *Index tables*: n tables, $n \geq 0$, storing index data derived from the user table.
(*iii*) *Meta table*: a single table for each database and each relational access method, storing $O(1)$ rows for each instance of an index.

The stored data is called *user data*, *index data*, and *meta data*.

To illustrate the concept of relational access methods, Figure 8 presents the minimum bounding rectangle list (*MBR-List*), a very simple example for indexing two-dimensional polygons. The user table is given by the object-relational table *polygons* (Figure 8a), com-

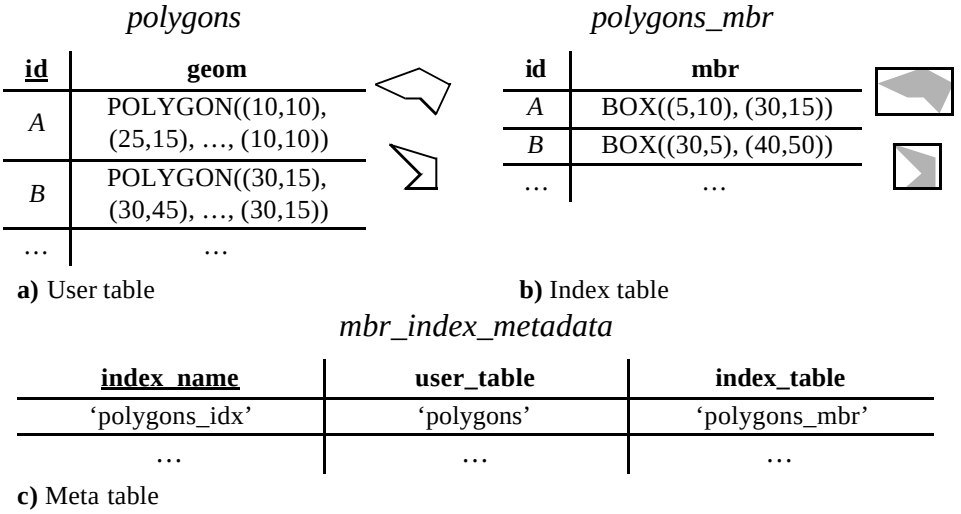


Figure 8: The *MBR-List*, a simple example for a relational access method

prising attributes for the polygon data type (*geom*) and the object identifier (*id*). Any spatial query can already be evaluated by sequentially scanning this user table. In order to speed up spatial selections, we decide to define an MBR-List *polygons_idx* on the user table. Thereby, an index table is created and populated (Figure 8b), assigning the minimum bounding rectangles (*mbr*) of each polygon to the foreign key *id.* Thus, the index table stores information purely derived from the user table. All schema objects belonging to the relational index, in particular the name of the index table, and other index parameters are stored in a global meta table (Figure 8c).

In order to support queries on the index tables, a relational access method can employ any built-in secondary indexes, including hash indexes, B+-trees, and bitmap indexes. Alternatively, index tables may be clustered by appropriate primary indexes. Consequently, the relational access method and the database system cooperate to maintain and retrieve the index data [DDSS95]. This basic approach of relational indexing has already been applied in many existing solutions, including *Linear Quadtrees* [TH81] [RS99] [FFS00] and *Relational R-trees* [RRSB 99] for spatial databases, *Relational X-trees* [BBKM99] for high-dimensional nearest-neighbor search, or inverted indexes for information retrieval on text documents [DDSS95].

4 Operations on Relational Access Methods

In the strict sense of the Definition1, the procedural code of an arbitrary block-oriented storage structure can immediately be transformed to a relational access method by replacing each invocation of the underlying block manager by an SQL-based DML operation[1]. Thus, the original procedural style of an index operation remains unchanged, whereas its I/O requests are now executed by a fully-fledged RDBMS. The object-relational database

1. E.g. we replace "blocks.get(*block_id*)" by "select * from blocks where id = *:block_id*".

server is thereby reduced to a plain block manager. In consequence, only a fraction of the existing functionality of the underlying database server is exploited. In this section, we define operations on relational access methods which maximize the architecture-awareness postulated in [JS99]. This can be achieved by using declarative operations.

4.1 Cursor-Bound Operations

In order to guarantee a better exploitation of the database infrastructure, we have to restrict the possible number of DML operations submitted from a procedural environment:

Definition 2 (*Cursor-Bound Operation*). A query or update operation on a relational access method is termed *cursor-bound*, iff the corresponding I/O requests on the index data can be performed by submitting $O(1)$ DML statements, i.e. by sequentially and concurrently opening in total $O(1)$ cursors provided by the underlying RDBMS.

Cursor-bound operations on relational access methods are largely bound to the declarative DML engine of the underlying RDBMS rather than to user-defined opaque code. Thus, the database server gains the responsibility for significant parts of the query and update semantics. Advantages of this approach include:

- **Declarative Semantics.** Large parts of a cursor-bound operation are expressed by using declarative SQL. By minimizing the procedural part and maximizing the declarative part of an operation, the formal verification of the semantics is simplified if we can rely on the given implementation of SQL to be sound and complete.
- **Query Optimization.** Whereas the database engine optimizes the execution of single, closed-form DML statements, a joint execution of multiple, independently submitted queries is very difficult to achieve [Sel88] [CD98] [BEKS00]. By using only a constant number of cursors, the RDBMS captures significant parts of the operational semantics at once. In particular, complex I/O operations including external sorting, duplicate elimination or grouping should be processed by the database engine, and not by a user-defined procedure.
- **Cursor Minimization.** The CPU cost of opening a variable number of cursors may become very high. For typical applications, the resulting overhead sums up to 30% of the total processing time [RMF+00]. In some experiments, we even reached barrier crossing cost of up to 75% for submitting a variable number of pre-parsed DML statements out of a stored procedure. For cursor-bound operations, the relatively high cost of opening and fetching multiple database cursors remains constant with respect to the complexity of the operation and the database size.

4.2 Cursor-Driven Operations

A very interesting case occurs if the potential result of a cursor-bound operation can be retrieved as the immediate output of a *single* cursor provided by the DBMS. Thus, the semantics is revealed to the database server at once in its full completeness:

Definition 3 (*Cursor-Driven Operation*). A cursor-bound operation on a relational access method is called *cursor-driven*, iff it can be divided into two consecutive phases:

(*i*) *Procedural phase*: In the first phase, index parameters are read from the meta tables. Query specifications are retrieved and data structures required for the actual query execution may be prepared by user-defined procedures and functions. Additional DML operations on user data or index data are not permitted.

(*ii*) *Declarative phase*: In the second phase, only a single DML statement is submitted to the ORDBMS, yielding a cursor on the final results of the index scan which requires no post-processing by user-defined procedures or functions.

Note that any cursor-driven operation is also cursor-bound, while all I/O requests on the index data are driven by a single declarative DML statement. The major advantage of cursor-driven operations is their smart integration into larger execution plans. After the completion of the procedural phase, the single DML statement can be executed with arbitrary groupings and aggregations, supplemented with additional predicates, or serve as a row source for joins. Furthermore, the integration into extensible indexing frameworks is facilitated, as the cursor opened in the declarative phase can be simply pipelined to the index scan routine. Note that the ability to implement cursor-bound and cursor-driven operations heavily relies on the expressive power of the underlying SQL interface, including the availability of recursive queries [Lib01].

The single DML statement submitted in the declarative phase may contain user-defined functions. The CPU cost of cursor-driven operations is significantly reduced, if the number of barrier crossings due to calls to user-defined functions is minimized [Kor99]. We can achieve this by preprocessing any required transformation, e.g. of a query specification, in the procedural phase and by bulk-binding the prepared data to the query statement with the help of transient collections. If such data structures become very large, a trade-off has to be achieved between the minimization of barrier crossings and the main-memory footprint of concurrent sessions. Splitting a single query into multiple cursor-driven operations can then be beneficial.

To pick up the *MBR-List* example of the previous section, Figure 9a shows a simple window query on the database of two-dimensional polygons, testing the exact geometry of each stored polygon for intersection with the query rectangle. In order to use the relational index as primary filter, the query has to be rewritten into the form of Figure 9b. An efficient execution plan for the rewritten query may first check the intersection with the stored bounding boxes, and refine the result by performing the equijoin with the *polygons* table. Note that the window query is a cursor-driven operation on the MBR-List, having an empty procedural phase. Therefore, the index-supported query can be easily embedded into a larger context as shown in Figure 9c. Already this small example shows that an object-relational wrapping of relational access methods is essential to control redundant data in the index tables and to avoid manual query rewriting. The usage of an extensible indexing framework preserves the physical independence of DML operations and enables the usual query optimization.

Although similarity queries or nearest neighbor queries („*return the k polygons closest to a query point wrt. to a given metric*“) can also be performed in a cursor-driven way by using the order-by clause together with a *top-k*-filter, the efficiency of this approach is rather questionable [CK97].

```
SELECT id FROM polygons
WHERE geom INTERSECTS BOX((0,0),(100,100));
```

a) Window query on the user table.

```
SELECT usr.id AS id FROM polygons usr, polygons_mbr idx
WHERE idx.mbr INTERSECTS BOX((0,0),(100,100))
AND idx.id = usr.id
AND usr.geom INTERSECTS BOX((0,0),(100,100));
```

b) Window query using the relational index as primary filter.

```
SELECT id FROM polygon_type
WHERE type = 'LAKE'
AND id IN (
  SELECT usr.id FROM polygons usr, polygons_mbr idx
  WHERE idx.mbr INTERSECTS BOX((0,0),(100,100))
  AND idx.id = usr.id
  AND usr.geom INTERSECTS BOX((0,0),(100,100))
);
```

c) Index-supported window subquery.

Figure 9: Window queries on two-dimensional polygons

5 Generic Schemes for Relational Indexing

As an immediate result of the relational storage of index data and meta data, a relational index is subject to the built-in transaction semantics, concurrency control, and recovery services of the underlying database system. In this section, we discuss the effectiveness and performance provided by the built-in services of the ORDBMS on relational access methods. For that purpose, we identify two generic schemes for the relational storage of index data, the *navigational* scheme and the *direct* scheme.

5.1 Navigational Scheme of Index Tables

Definition 4 (*Navigational Scheme*).

Let $P = (T, R_1, ..., R_n)$ be a relational access method on a data scheme T and index schemes $R_1, ..., R_n$. We call P *navigational* $\Leftrightarrow (\exists\, t \subseteq T)\ (\exists\, r_i \subseteq R_i, 1 \le i \le n)$: at least one $\rho \in r_i$ is associated with rows $\{\tau_1, ..., \tau_m\} \subseteq t$ and $m > 1$.

Therefore, a row in an index table of a navigational index may logically represent many objects stored in the user table. This is typically the case for hierarchical structures that are mapped to a relational schema. Consequently, an index table contains data that is recursive-

ly traversed at query time in order to determine the resulting tuples. Examples for the navigational scheme include the *Oracle Spatial R-tree* [RRSB99] and the *Relational X-tree* [BBKM99] which store the nodes of a tree directory in a flat table. To implement a navigational query as a cursor-bound operation, a recursive version of SQL like SQL:1999 [SQL99] [EM99] is required.

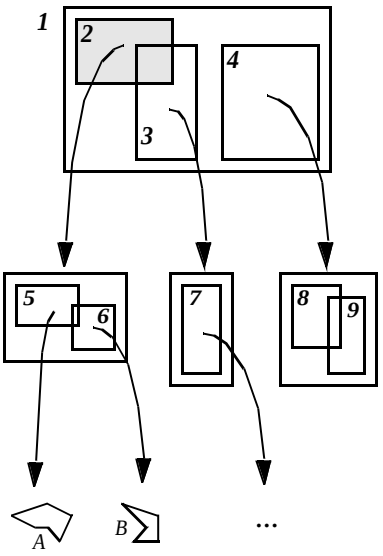
Although the navigational scheme offers a straightforward way to simulate any hierarchical index structure on top of a relational data model, it suffers from the fact that navigational data is locked like user data. As two-phase locking on index tables is too restrictive, the possible level of concurrency is unnecessarily decreased. For example, uncommitted node splits in a hierarchical directory may lock entire subtrees against concurrent updates. Built-in indexes solve this problem by committing structural modifications separately from content changes [KB95]. Unfortunately, this approach is not feasible on the SQL layer without braking up the user transaction. A similar overhead exists with logging, as atomic actions on navigational data, e.g. node splits, are not required to be rolled back in order to keep the index tables consistent with the data table. Therefore, relational access methods implementing the navigational scheme are only well suited for read-only or single-user environments.

5.2 Relational R-trees – An Example for the Navigational Scheme

We illustrate the properties and drawbacks of the navigational scheme by the example of *Relational R-trees*, like they have been used by the Oracle developers Ravi Kanth et al. [RRSB99]. Figure 10 depicts a hierarchical R-tree along with a possible relational mapping (*page_id, page_lev, son_id, son_mbr*). The column *page_id* contains the logical page identifier, while *page_lev* denotes its level in the tree. Thereby, 0 marks the level of the data objects, and 1 marks the leaf level of the directory. The attribute *son_id* contains the *page_id* of the connected entry, while *son_mbr* stores its minimum bounding rectangle. Thus, *page_id* and *son_id* together comprise the primary key. In our example, the logical page ***2*** represents a partition of the data space which contains the polygons *A* and *B*. The corresponding index row (***1***, 2, ***2***,…) is therefore logically associated with the rows (*A*, …) and (*B*,…) in the *polygons* user table (cf. Figure 8). Thus, the Relational R-tree implements the navigational scheme of relational access methods.

The severe overhead of the navigational scheme already becomes obvious if a transaction inserts a new polygon, and subsequently enlarges the bounding box of a node, e.g. of the root node. Due to the common two-phase locking, this transaction will hold an exclusive lock on the row (***ROOT***, 3, ***1***,…) until commit or rollback. During this time, no concurrent transaction can insert polygons that induce an enlargement of the root region. The database server has to guarantee non-blocking reads [Ora99c] to support at least concurrent queries on the Relational R-tree index.

To support the navigation through the R-tree table at query time, a built-in index can be created on the *page_id* column. Alternatively, the schema can be transformed to NF^2 (non-first normal form), where *page_id* alone represents the primary key, and a collection of (*son_id*, *son_mbr*) pairs is stored with each row. In this case, the static storage location of each tuple can be used as *page_id*, avoiding the necessity of a built-in index. A cursor-driven primary filter for a window query using recursive SQL is shown in Figure 11. We expect that future implementations of the SQL:1999 statement yield a depth-first traversal which is already hard-wired into the existing CONNECTBY clause of the Oracle server. The ef-



a) Hierarchical directory

polygons_rtree

pageid	page_lev	son_id	son_mbr
ROOT	3	1	BOX((0,0), (200,120))
1	2	2	BOX((0,0), (80,60))
1	2	3	BOX((60,20), (100,120))
1	2	4	BOX((140,20), (200,120))
2	1	5	…
2	1	6	…
5	0	A	…
6	0	B	…
…	…	…	…

b) Relational index table

Figure 10: Relational mapping of an R-tree directory

fectiveness of cursor-driven operations is illustrated by the fact that the depicted statements already comprise the complete, pipelined query processing on the R-tree index. If the low concurrency of the Relational R-tree is acceptable, the relational mapping opens up a wide range of potential improvements. We have developed and evaluated various extensions to the presented concept[1]:

- **Variable Fanout.** Due to the relational mapping, we are basically free to allow an individual fanout for each tree node. Similar to the concept of supernodes for high-dimensional indexing [BKK96], larger nodes could be easily allocated, e.g. if the contained geometries show a very high overlap or are almost equal. Thus, splitting such pages would not improve the spatial clustering. Instead, page splits could be triggered by measuring the clustering quality with a proximity measure similar to [KF92]. Especially for CAD databases, where many variants of the same parts occupy almost identical regions of the data space, this approach can be beneficial.
- **Page Clustering.** In order to achieve a good clustering among the entries of each tree node, a built-in primary index can be defined on the *page_id* column. For bulk-loads of Relational R-trees, the clustering can be further improved by carefully choosing the page identifiers: by assigning linearly ordered *page_ids* corresponding to a breadth-first traversal of the tree, a sibling clustering of nodes [KC98] can be very easily achieved.
- **Positive Pruning.** By ordering the *page_ids* according to a depth-first tree traversal, a hierarchical clustering of the R-tree nodes is materialized in the primary index. In consequence, the page identifiers of any subtree form a consecutive range. Similarly, if the

1. We refer the reader to [Bra00] for detailed descriptions.

```
WITH RECURSIVE tree_traversal (page_lev, son_id, son_mbr) AS (
  SELECT page_lev, son_id, son_mbr FROM polygons_rtree
  WHERE page_id = ROOT
  UNION ALL
  SELECT next.page_lev, next.son_id, next.son_mbr
  FROM tree_traversal prior, polygons_rtree next
  WHERE prior.page_mbr INTERSECTS BOX((0,0),(100,100))
  AND prior.son_id = next.page_id
)                                                    //declarative tree traversal
SELECT son_id AS id
FROM tree_traversal
WHERE page_lev = 0;                                  //select data objects
```

a) Recursive window query on a Relational R-tree using SQL:1999.

```
SELECT son_id AS id FROM polygons_rtree
WHERE page_lev = 0                                   //select data object
START WITH page_id = ROOT
CONNECT BY
  PRIOR son_mbr INTERSECTS BOX((0,0),(100,100))
  AND PRIOR son_id = page_id;                        //declarative tree traversal
```

b) Recursive window query on a Relational R-tree using Oracle SQL.

Figure 11: Cursor-driven window query on a Relational R-tree

leaf pages are hierarchically clustered in a separate B+-tree, a single range query on the *page_id* column yields a blocked output of all data objects stored in any arbitrary subtree of the R-directory. Thus, the recursive tree traversal below a node completely covered by the query region can be replaced by an efficient range scan on the leaf table. Consequently, the tree traversal is not only pruned for all-negative nodes (if no intersection of the node region with the query region is detected), but also for all-positives (the node region is completely covered by the query region). Moreover, heuristics to prune already largely covered nodes can also be very beneficial.

5.3 Direct Scheme of Index Tables

Definition 5 (*Direct Scheme*).

Let $P = (T, R_1, ..., R_n)$ be a relational access method on a data scheme T and index schemes $R_1, ..., R_n$. We call P *direct* $\Leftrightarrow (\forall\, t \subseteq T)\ (\forall\, r_i \subseteq R_i, 1 \le i \le n)$: each $\rho \in r_i$ is associated with a single row $\tau \in t$.

In consequence, for a relational access method of the direct scheme, each row in the user table is directly mapped to a set of rows in the index tables. Inversely, each row in an index table exclusively belongs to a single row in the user table. In order to support queries, the

index table is organized by a built-in index, e.g. a B+-tree. Examples for the direct scheme include our *MBR-List* (cf. Figure 8), the *Linear Quadtree* [Sam90], the one-dimensional *Relational Interval Tree* [KPS00] and its optimization for interval sequences and multidimensional queries [KPS01].

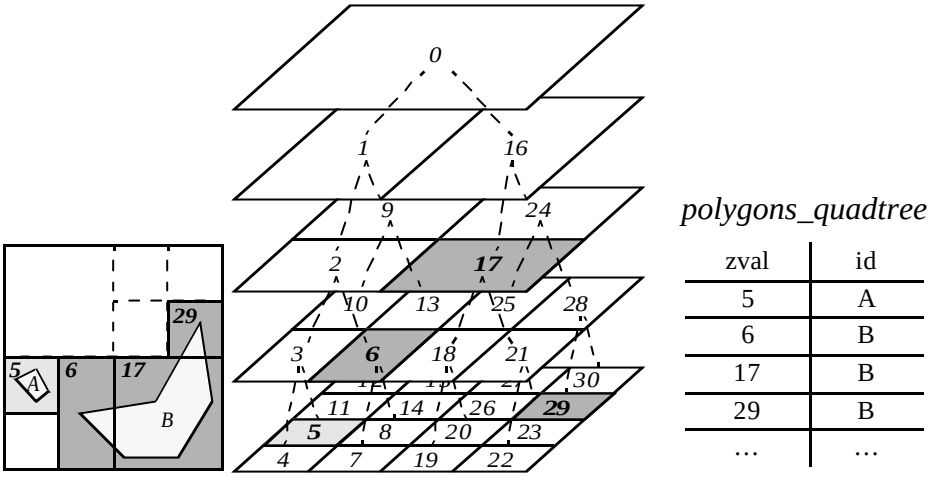
The drawbacks of the navigational scheme with respect to concurrency control and recovery are not shared by the direct scheme, as row-based locking and logging on the index tables can be performed on the granularity of single rows in the user tables. For example, an update of a single row *r* in the user table requires only the synchronization of index rows exclusively assigned to *r*. As the acquired locks are restricted to *r* and its exclusive entries in the index tables, they do not unnecessarily block concurrent operations on other user rows. In contrast to navigational indexes, the direct scheme inherits the high concurrency and efficient recovery of built-in tables and indexes.

5.4 Linear Quadtrees – An Example for the Direct Scheme

A paradigmatic example for a spatial access method implementing the direct scheme is the *Linear Quadtree* [Sam90]. Several variants of this well-known concept have been proposed for stand-alone balanced trees [TH81] [OM84] [Bay96], for object-oriented database systems [Ore 86] [OM 88] [GR 94], and as relational access methods [Wan91] [IBM98] [Ora99b] [RS99] [FFS00]. In this subsection, we present the basic idea of the Linear Quadtree according to the in-depth discussion of Freytag, Flasza and Stillger [FFS00].

The Linear Quadtree organizes the multidimensional data space by a regular grid. Any spatial object is approximated by a set of *tiles*. Among the many possible one-dimensional embeddings of a grid approximation, the *Z-order* is one of the most popular [Güt94]. The corresponding index representation of a spatial object comprises a set of *Z-tiles* which is computed by recursively bipartitioning the multidimensional grid. By numbering the Z-tiles of the data space according to a depth-first recursion into this partitioning, any set of Z-tiles can be represented by a set of linear values. Note that thereby redundancy is introduced to approximate spatially extended data. Figure 12 depicts some Z-tiles on a two-dimensional grid along with their linear values. The linear values of the Z-tiles of each spatial object can be stored in an index table obeying the schema (*zval*, *id*), where both columns comprise the primary key. This relational mapping implements the direct scheme, as each row in the index table exclusively belongs to a single data object. The linear ordering positions each Z-tile of an object on its own row in the index table. Thus, if a specific row in the user table *polygons* is updated, e.g. (*B*,…), only the rows (***6***, *B*), (***17***, *B*), and (***29***, *B*) in the index table are affected, causing no problems with respect to the native two-phase locking.

In order to process spatial selection on the Linear Quadtree, the query region is also required to be decomposed to a set of Z-tiles. We call the corresponding function *ZDecompose*. For each resulting linear value *zval*, the intersecting tiles have to be extracted from the index table. Due to the Z-order, all intersecting tiles having the same or a smaller size than the tile represented by *zval* occupy the range *ZLowerHull*(*zval*) = [*zval*, *ZHi*(*zval*)] which can be easily computed [FFS00]. In the example of Figure 12, we obtain *ZLowerHull*(17)=[17,23]. In a similar way, we also compute *ZUpperHull*(*zval*), the set of all larger intersecting tiles. As in the case of *ZUpperHull*(17)={0,16} the corresponding linear values usually form no consecutive range. To find all intersecting tiles for



a) Regular grid **b)** Recursive partitioning **c)** Index table

Figure 12: Relational mapping of a Linear Quadtree

a given *zval*, a range scan on the index table is performed with *ZLowerHull*(*zval*) and multiple exact match queries are executed for *ZUpperHull*(*zval*). These queries are optimally supported by a built-in B+-tree on the *zval* column. Figure 13 depicts the complete cursor-driven window query on an instance of the Linear Quadtree using SQL:1999. Alternatively, the transient rowsets generated by the functions *ZDecompose* and *ZUpperHull* can be precomputed in the procedural phase for all Z-tiles of the query box and passed to the SQL layer in one step by using bind variables. This approach reduces the overhead of barrier crossings between the declarative and procedural environments to a minimum.

6 Conclusions

In this paper, we presented the concept of relational access methods which employ the infrastructure and functionality of existing object-relational database systems to provide efficient execution plans for the evaluation of user-defined predicates. We introduced cursor-bound and cursor-driven operations to maximize the achievable declarativity, usability

```
SELECT DISTINCT idx.id                                  //select data object
FROM    polygons_quadtree idx,
        TABLE(ZDecompose(BOX((0,0),(100,100)))) tiles,
        TABLE(ZUpperHull(tiles.zval)) uh
WHERE  (idx.zval BETWEEN tiles.zval AND ZHi(tiles.zval))
        OR (idx.zval = uh.zval);
```

Figure 13: Cursor-driven window query on a Linear Quadtree

and performance of operations. We identified two generic schemes for the relational mapping of index data, each having different properties with respect to the built-in locking and logging mechanisms of the underlying database engine: Whereas the *navigational* scheme seems only appropriate for single-user or read-only databases, the *direct* scheme fully preserves the effectivity and efficiency of built-in transactions, concurrency control, and recovery services. The presented concepts have been illustrated by three spatial examples: The *MBR-List*, a trivial relational access method for demonstration purposes, along with the *Relational R-tree* and the *Linear Quadtree*, two fully-fledged spatial access methods implementing the navigational and the direct scheme, respectively.

In our future work we plan to investigate whether there are generic patterns to develop a relational indexing scheme for any given index structure. Again, a careful analysis of the potentials and the overhead of relational data management is a major point of interest.

Acknowledgements. We would like to thank the anonymous referees for their constructive and helpful comments.

References

[Aok 98] Aoki P. M.: *Generalizing "Search" in Generalized Search Trees*. Proc. 14th Int. Conf. on Data Engineering (ICDE): 380-389, 1998.

[AV 96] Arge L., Vitter J. S.: *Optimal Dynamic Interval Management in External Memory.* Proc. 37th Annual Symp. on Foundations of Computer Science: 560-569, 1996.

[Bay 96] Bayer R.: *The Universal B-Tree for multidimensional Indexing*. Technical University of Munich, TUM-I9637, 1996.

[BBKM 99] Berchtold S., Böhm C., Kriegel H.-P., Michel U.: *Implementation of Multidimensional Index Structures for Knowledge Discovery in Relational Databases*. Proc. 1st Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK), LNCS 1676: 261-270, 1999.

[BEKS 00] Braunmüller B., Ester M., Kriegel H.-P., Sander J.: *Efficiently Supporting Multiple Similarity Queries for Mining in Metric Databases.* Proc. 16th Int. Conf. on Data Engineering (ICDE): 256-267, 2000.

[BKK 96] Berchtold S., Keim D. A., Kriegel H.-P.: *The X-tree: An Index Structure for High-Dimensional Data*. Proc. 22nd Int. Conf. on Very Large Databases (VLDB): 28-39, 1996.

[Bra 00] Braun C.: *Development and Evaluation of R-Trees for Object-Relational Database Systems* (in german). Diploma Thesis, University of Munich, 2000.

[Bro 01] Brown P.: *Object-Relational Database Development – A Plumber's Guide*. Informix Press, Menlo Park, CA, 2001.

[BSSJ 99] Bliujute R., Saltenis S., Slivinskas G., Jensen C.S.: *Developing a DataBlade for a New Index*. Proc. 15th Int. Conf. on Data Engineering (ICDE): 314-323, 1999.

[CCF+ 99] Chen W., Chow J.-H., Fuh Y.-C., Grandbois J., Jou M., Mattos N., Tran B., Wang Y.: *High Level Indexing of User-Defined Types*. Proc. 25th Int. Conf. on Very Large Databases (VLDB): 554-564, 1999.

[CD 98] Chen F.-C. F., Dunham M. H.: *Common Subexpression Processing in Multiple-Query Processing*. IEEE Trans. on Knowledge and Data Engineering, 10(3): 493-499, 1998.

[CK 97] Michael J. Carey, Donald Kossmann: *On Saying "Enough Already!" in SQL.* Proc. ACM SIGMOD Int. Conf. on Management of Data: 219-230, 1997.

[DDSS 95] DeFazio S., Daoud A., Smith L. A., Srinivasan J.: *Integrating IR and RDBMS Using Cooperative Indexing*. Proc. 18th ACM SIGIR Conference on Research and Development in Information Retrieval: 84-92, 1995.

[DKO+ 85] DeWitt D. J., Katz R. H., Olken F., Shapiro L. D., Stonebraker M., Wood D. A.: *Implementation Techniques for Main Memory Database Systems*. Proc. ACM SIGMOD Int. Conf. on Management of Data: 1-8, 1984.

[Ede 80] Edelsbrunner H.: *Dynamic Rectangle Intersection Searching*. Institute for Information Processing Report 47, Technical University of Graz, Austria, 1980.

[EM 99] Eisenberg A., Melton J.: *SQL:1999, formerly known as SQL3*. ACM SIGMOD Record, 28(1): 131-138, 1999.

[FFS 00] Freytag J.-C., Flasza M., Stillger M.: *Implementing Geospatial Operations in an Object-Relational Database System*. Proc. 12th Int. Conf. on Scientific and Statistical Database Management (SSDBM): 209-219, 2000.

[GHJV 95] Gamma E., Helm R., Johnson R., Vlissides J.: *Design Patterns*. Addison Wesley Longman, Boston, MA, 1995.

[GR 94] Gaede V., Riekert W.-F.: *Spatial Access Methods and Query Processing in the Object-Oriented GIS GODOT*. Proc. AGDM Workshop, Geodetic Commission, 1994.

[GS 92] Garcia-Molina H., Salem K.: *Main Memory Database Systems: An Overview*. IEEE Trans. on Knowledge and Data Engineering 4(6): 509-516, 1992.

[Güt94] Güting R. H.: *An Introduction to Spatial Database Systems*. VLDB Journal , 3(4): 357-399, 1994.

[HNP 95] Hellerstein J. M., Naughton J. F., Pfeffer A.: *Generalized Search Trees for Database Systems*. Proc. 21st Int. Conf. on Very Large Databases: 562-573, 1995.

[IBM 98] IBM Corp.: *IBM DB2 Spatial Extender Administration Guide and Reference, Version 2.1.1*. Armonk, NY, 1998.

[IBM 99] IBM Corp.: *IBM DB2 Universal Database Application Development Guide, Version 6*. Armonk, NY, 1999.

[Inf 98] Informix Software, Inc.: *DataBlade Developers Kit User's Guide, Version 3.4*. Menlo Park, CA, 1998.

[Inf 99] Informix Software, Inc.: *Informix R-Tree Index User's Guide, Version9.2*. Menlo Park, CA, 1999.

[JS 99] Jensen C. S., Snodgrass R. T.: *Temporal Data Management*. IEEE Trans. on Knowledge and Data Engineering 11(1): 36-44, 1999.

[KB 95] Kornacker M., Banks D.: *High-Concurrency Locking in R-Trees*. Proc. 21st Int. Conf. on Very Large Databases (VLDB): 134-145, 1995.

[KC 98] Kim K., Cha S. K.: *Sibling Clustering of Tree-based Spatial Indexes for Efficient Spatial Query Processing*. Proc. ACM CIKM Int. Conf. on Information and Knowledge Management: 398-405, 1998.

[KF 92] Kamel I., Faloutsos C.: *Parallel R-trees*. Proc. ACM SIGMOD Int. Conf. on Management of Data: 195-204, 1992.

[KMH 97] Kornacker M., Mohan C., Hellerstein J. M.: *Concurrency Control in Generalized Search Trees*. Proc. ACM SIGMOD Int. Conf. on Management of Data: 62-72, 1997.

[KMPS01a] Kriegel H.-P., Müller A., Pötke M., Seidl T.: *DIVE: Database Integration for Virtual Engineering* (Demo). Demo Proc. 17th Int. Conf. on Data Engineering (ICDE): 15-16, 2001.

[KMPS01b] Kriegel H.-P., Müller A., Pötke M., Seidl T.: *Spatial Data Management for Computer Aided Design* (Demo). Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.

[Kor 99] Kornacker M.: *High-Performance Extensible Indexing*. Proc. 25th Int. Conf. on Very Large Databases (VLDB): 699-708, 1999.

[KP 92] Keim D. A., Prawirohardjo E. S.: *Datenbankmaschinen – Performanz durch Parallelität*. Reihe Informatik 86, BI Wissenschaftsverlag, Mannheim, 1992.

[KPS 00] Kriegel H.-P., Pötke M., Seidl T.: *Managing Intervals Efficiently in Object-Relational Databases*. Proc. 26th Int. Conf. on Very Large Databases (VLDB): 407-418, 2000.

[KPS 01] Kriegel H.-P., Pötke M., Seidl T.: *Interval Sequences: An Object-Relational Approach to Manage Spatial Data*. Proc. 7th Int. Symposium on Spatial and Temporal Databases (SSTD), LNCS 2121: 481-501, 2001.

[Lib 01] Libkin L.: *Expressive Power of SQL*. Proc. 8th Int. Conf. on Database Theory (ICDT): 1-21, 2001.

[OM 84] Orenstein J. A., Merrett T. H.: *A Class of Data Structures for Associative Searching*. Proc. 3rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS): 181-190, 1984.

[OM 88] Orenstein J. A., Manola F. A.: *PROBE Spatial Data Modeling and Query Processing in an Image Database Application*. IEEE Transactions on Software Engineering, 14(5): 611-629, 1988.

[Ora 99a] Oracle Corp.: *Oracle8i Data Cartridge Developer's Guide, Release 2 (8.1.6)*. Redwood Shores, CA, 1999.

[Ora 99b] Oracle Corp.: *Oracle Spatial User's Guide and Reference, Release 8.1.6*. Redwood Shores, CA, 1999.

[Ora 99c] Oracle Corp.: *Oracle8i Concepts, Release 8.1.6*. Redwood Shores, CA, 1999.

[Ora 00] Oracle Corp.: *Oracle8i Appliance – An Oracle White Paper*. Redwood Shores, CA, 2000.

[Ore 86] Orenstein J.A.: *Spatial Query Processing in an Object-Oriented Database System*. Proc. ACM SIGMOD Int. Conf. on Management of Data: 326-336, 1986.

[PS 93] Preparata F. P., Shamos M. I.: *Computational Geometry: An Introduction*. 5th ed., Springer, 1993.

[RMF+ 00] Ramsak F., Markl V., Fenk R., Zirkel M., Elhardt K., Bayer R.: *Integrating the UB-Tree into a Database System Kernel*. Proc. 26th Int. Conf. on Very Large Databases (VLDB): 263-272, 2000.

[RRSB 99] Ravi Kanth K. V., Ravada S., Sharma J., Banerjee J.: *Indexing Medium-dimensionality Data in Oracle*. Proc. ACM SIGMOD Int. Conf. on Management of Data: 521-522, 1999.

[RS 99] Ravada S., Sharma J.: *Oracle8i Spatial: Experiences with Extensible Databases*. Proc. 6th Int. Symp. on Large Spatial Databases (SSD), LNCS 1651: 355-359, 1999.

[Sam 90] Samet H.: *Applications of Spatial Data Structures*. Addison Wesley Longman, Boston, MA, 1990.

[Sel 88] Sellis T. K.: *Multiple-Query Optimization*. ACM Transactions on Database Systems (TODS), 13(1): 23-52, 1988.

[SDF+ 00] Jagannathan Srinivasan, Souripriya Das, Chuck Freiwald, Eugene Inseok Chong, Mahesh Jagannath, Aravind Yalamanchi, Ramkumar Krishnan, Anh-Tuan Tran, Samuel DeFazio, Jayanta Banerjee: *Oracle8i Index-Organized Table and Its Application to New Domains*. Proc. 26th Int. Conf. on Very Large Databases (VLDB): 285-296, 2000.

[SMS+ 00] Srinivasan J., Murthy R., Sundara S., Agarwal N., DeFazio S.: *Extensible Indexing: A Framework for Integrating Domain-Specific Indexing Schemes into Oracle8i*. Proc. 16th Int. Conf. on Data Engineering (ICDE): 91-100, 2000.

[SQL 92] American National Standards Institute: *ANSI X3.135-1992/ISO 9075-1992 (SQL-92)*. New York, NY, 1992.

[SQL 99] American National Standards Institute: *ANSI/ISO/IEC 9075-1999 (SQL:1999, Parts 1-5)*. New York, NY, 1999.

[Sto 86] Stonebraker M.: *Inclusion of New Types in Relational Data Base Systems*. Proc. 2nd Int. Conf. on Data Engineering (ICDE): 262-269, 1986.

[TH 81] Tropf H., Herzog H.: *Multidimensional Range Search in Dynamically Balanced Trees*. Angewandte Informatik, 81(2): 71-77, 1981.

[Wan 91] Wang F.: *Relational-Linear Quadtree Approach for Two-Dimensional Spatial Representation and Manipulation*. IEEE Trans. on Knowledge and Data Engineering (TKDE) 3(1): 118-122, 1991.

Multidimensional Mapping and Indexing of XML

Michael G. Bauer, Frank Ramsak, Rudolf Bayer
Institut für Informatik, TU München
Boltzmannstr. 3, D-85747 Graching bei München, Germany
{bauermi, ramsak, bayer}@in.tum.de

Abstract: We propose a multidimensional approach to store XML data in relational database systems. In contrast to other efforts we suggest a solution to the problem using established database technology. We present a multidimensional mapping scheme for XML and also thoroughly study the impact of established and commercially available multidimensional index structures (compound B-Trees and UB-Trees) on the performance of the mapping scheme. In addition, we compare our multidimensional mapping to other known mapping schemes. While studying the performance we have identified projection and selection to be fundamental parts of a typical query on XML documents. Our measurements show that projection and selection are orthogonal and require special multidimensional index support to be processed efficiently.

1 Introduction

XML is widely seen as the lingua franca of the Internet. Originally designed to become the successor of HTML, XML has found its way into many unexpected parts of applications, ranging from simple formats for data exchange to archiving data in XML. With the growing need to deal with large collections of XML documents as well as with the rapid increase in the document sizes there is a strong demand to store and query XML data in databases. This ranges from the development of new, efficient index structures for XML to mapping schemes for XML to relational and object-oriented database systems. As relational database management systems (RDBMS) hold the largest market share there was and still is intensive research going on to efficiently store XML in these systems. Several mapping schemes have been proposed in the literature [FK99, CSF+01] but to our knowledge there has never been an extensive analysis of a mapping scheme in conjunction with index structures.

In our work we discuss a multidimensional approach for indexing XML. We propose a multidimensional mapping scheme for XML to relational DBMS and discuss the performance of UB-Trees and compound B-Trees for the indexing of this mapping scheme.

The rest of the paper is structured as follows. At first we motivate the problem of XML indexing (Section 2) in RDBMS. Then we present a modelling of the XML document in a multidimensional universe (Section 3). We describe an implementation of our multidimensional approach by using Multidimensional Hierarchical Clustering (MHC) and propose a database schema and a technique for query rewriting on this schema (Section 4). We

also present detailed performance experiments using the UB-Tree and various compound B-Trees as multidimensional index structures in Section 5. We conclude the paper with a short summary in Section 7.

2 The Problem: Storing XML in RDBMSs

Many approaches have been made to store XML in relational database systems. All approaches use a similar concept though. First the XML document is split into parts of a previously defined granularity. These parts are stored in the RDBMS. Queries on the XML documents which are written in an XML query language have to be rewritten to SQL before being processed by the RDBMS. The results of the SQL queries are documents. The projection is either done via methods like XSLT or the projection results are assembled directly from the database. Our approach especially takes care of both cases.

Besides storing XML data, querying large amounts of XML data is a challenging problem. Due to its graph-like nature the classical set oriented query languages in general are not powerful enough. After several proposals for query languages (mainly from the fast evolving field of semistrucutred data) the W3C started a working group to formulate a query language for XML. The current proposal XQuery though is not yet a recommendation of the W3C. Throughout this paper we instead use a form of Pseudo-SQL to present queries on XML documents. We have chosen Pseudo-SQL as it is more similar to SQL dialects available in RDBMS. It is also easier to point out important requirements when querying mapped XML data with SQL. We do not consider Pseudo-SQL to be a full-fledged query language for XML, although it is of course possible to formulate the queries of this paper in XQuery (or XPath) without any loss of semantics. Due to the above restrictions the way vice versa is apparently not true.

We have identified two fundamental parts of a query for XML. Consider the following query in Pseudo-SQL which should retrieve the value of a tag (tag1) in a document containing the tag with value ABC: **select <tag1> from xmlbase where tag2='ABC';**. This query can be (similarly to queries in the relational case) split up into two steps. The selection part identifies the document(s) which contain(s) the pattern matching a predicate s. The projection part in contrast returns only those part(s) of the document(s) which are stated right after the **select** statement as a set of paths. The mentioned problems can be defined more formally:

Definition 1 (selection problem) *Let X be a set of XML documents stored in a database where each document is described by a persistant unique identifier Id. The selection problem is defined as returning Id for those documents where the predicate s from the query is evaluated to $true$.*

Definition 2 (projection problem) *Let I be a set of persistant unique identifiers and L be a list of path expressions in a projection list. The projection problem is defined as returning the content of those paths from the documents referenced by I that fullfill the expressions in L.*

When talking about the projection problem we sometimes use the term ”reconstruction of (parts of) the document”. We refer to the fact that the desired document is completely or partially assembled from the database into its original state.

In our approach we do not tackle only one of the above mentioned problems (either selection or projection) but both.

One has to be aware of the fact that the projection in the case of XML documents is fundamentally different from the relational world. The operations in the relational algebra deal with tuples and sets as the basic units. Tuples in RDBMSs are typically small compared to the size of an XML document, therefore tuples are normally retrieved as a whole during the selection process and the projection always processes the tuples that resulted from the selection. For effi ciently answering queries in the relational world it is therefore suffi cient to only speed-up the selection. For XML documents the scenario is slightly different as the granularity of an XML document can be seen on different levels. A very rough granularity is based on documents (which are identifi ed by a persistent unique identifi er). Tags as the building block of XML documents are another level of granularity, a very fi ne granularity would be based on words or letters. Depending on the storage of XML in the RDBMS the projection works on a completely different position of the XML document than the selection. The consequence is that an index that is suitable for the selection is rarely suitable to speed up projection as well. We will see later that selection and projection can even be orthogonal and require very specialized index support.

3 XML - A Multidimensional Model

Paths are a fundamental feature of XML. Many approaches to speed up queries therefore concentrate on the effi cient indexing of paths and path expressions. When discussing paths it is also important to note the order of paths which is inherent in the XML documents. The ordering of XML documents is a very important aspect especially in the case of indexing. The DTD of an XML document already defi nes the ordering of the tags in the document. This is especially important when tags with the same substructure occur several times but with different data.

Example 1

```
<author><FN>Michael</FN><LN>Bauer</LN></author>
<author><FN>Rudolf</FN><LN>Bayer</LN></author>
```

Order is also important for query languages. When the above example (a fragment from a larger document) is queried with the predicate **author/FN = Michael and author/LN = Bayer** the semantics of the predicate is not clear at fi rst hand. In the document fragment the paths to the values are the same but the structure and the order of the paths is of great importance as it acts as a grouping feature that is very relevant for queries. The above predicate can be reformulated so that the semantics is clear and the two parts of the query have to be valid in the same subtree. A correct version of the above predicate in XPath returning the above fragment is formulated as **author[FN = ”Michael” and LN =**

"Bauer"]. To be able to evaluate such a predicate it is apparently necessary to preserve the document structure for the stored document.

Finally, ordering might have a severe impact on the performance of answering queries. In every database instance the data is stored in a certain physical order on external storage devices. Clustering the stored data in document order usually can be achieved, nevertheless an order independent of the document order might be more useful for certain queries. Nevertheless, the original order of the stored document has to be preserved in some way, otherwise it is impossible to reconstruct the document in the same order as it was originally available.

In the following we propose three building blocks of XML documents.

Paths: The notation of paths is a basic concept of XML and query languages for XML. Due to the graph-like nature of XML it is necessary to preserve path information and order for query processing.

Values: We define values as the content of XML tags. Our proposed scheme can deal with both, data-centric and document-centric XML. Attributes in XML are modelled by using the @notation in the paths as known from XPath.

Document Identifiers: Document identifiers group paths for one document and are the results in the above mentioned selection. We assume that this identifier is available from the XML data itself. If this is not the case it can be easily computed when the document is processed before it is inserted into the database.

Summing up our results we can define a set of XML documents which are stored in our database as follows.

Definition 3 *Let P be a set of paths, $<_{path}$ the order of the paths, V a set of values and id a document identifier.*

$$\mathit{XMLdocs} = \bigcup \{(P, <_{path}, V, id) | id = docid\}$$

For better visualization the three dimensions can be presented as a cube (Figure 1). In this three dimensional model we can now easily answer queries for both the selection and the projection problem as follows.

Both selection and projection restrict the threedimensional universe in two dimensions. The input for evaluation of the XPath predicate of the selection is one (or more) path expressions and one (or more) values which correspond to the path expression and form a predicate. The output of the selection is a set of document identifiers which match path expression(s) and value(s).

In the projection the document identifiers and the path expressions are the input of the query. The results are values. For better readability and post-processing capabilities the document identifiers and the output paths are sometimes returned as well.

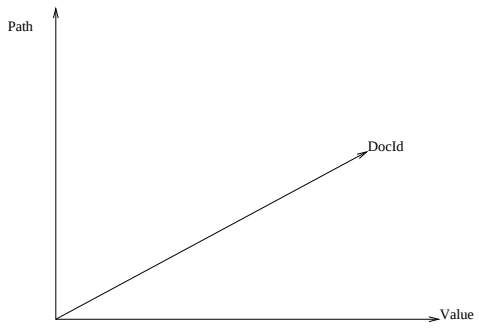


Figure 1: XML documents in a three dimensional cube

4 Implementation

4.1 Multidimensional Hierarchical Clustering

Multidimensional Hierarchical Clustering (MHC) is a technique that was originally developed for data warehousing applications [MRB99]. In data warehousing MHC is used to cluster data with respect to multiple hierarchical dimensions. MHC enables us to find a compact and order preserving representation for paths in XML documents. It also fixes the problem of long equal prefixes which has already been addressed in work on special index structures for XML [CSF+01]. We briefly describe the technique in an overview, show its application in our context of XML indexing, how we use MHC to preserve order and how to store paths in a compact form.

It is well known by now that XML documents form a hierarchy. We assume that each XML document has a maximal hierarchy of depth h leading to an overall of $h + 1$ levels in each document. We use a slightly different tree representation as it is usually common. Identical paths which occur several times within a subtree are rewritten by using repetition numbers to preserve uniqueness and order. The paths from example 1 are rewritten to <**author**>**[1]**<**FN**>**[1],** <**author**>**[1]**<**LN**>**[1],**
<**author**>**[2]**<**FN**>**[1],** <**author**>**[2]**<**LN**>**[1]** .

In the following we treat the repetition numbers as an own hierarchy level. The set of levels is ordered according to the document structure. Each hierarchy level i is a set of sets L ($L = \bigcup_{i=0}^{n} L_i$), where each set L_i consists of nodes m_k^i. L_0 is defined to be the root level. Due to the hierarchical nature every member m_k^i has a (varying) number of children. A function ord_m defines a numbering scheme for the children of m_k^i and assigns each child of m_k^i a number between 1 and the total number of children of m_k^i, i.e., $ord_m : children(m) \longrightarrow \{1, \ldots, |children(m)|\}$. The function ord_m needs to be order preservering so that nodes are sequentially numbered according to document order. We call each element of $\{1, \ldots, |children(m)|\}$ a surrogate of a node. To encode paths through the hierarchy we introduce the concept of compound surrogates. A compound

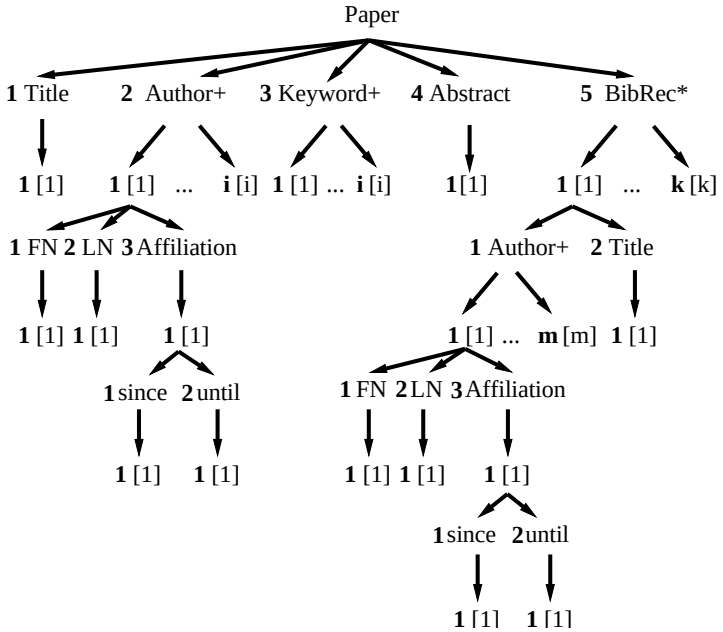


Figure 2: XML document hierarchy and MHC surrogates

surrogate is formed by recursively concatenating the compound surrogate of the father node with the surrogate of the current node. More formally the compound surrogate for a node is defined as follows:

$$cs(m^i) = \begin{cases} ord_{father(m^i)}(m^i), & \text{if } i = 1 \\ cs(father(m^i)) \circ ord_{father(m^i)}(m^i), & otherwise \end{cases}$$

Example 2 *Using Figure 2 the path* `<author>[1]<FN>[1]` *is transformed into the compound surrogate 2.1.1.1*

Compound surrogates can be very efficiently stored in a compact binary representation. The upper limit for each surrogate can be calculated by the formula

$$surrogate(i) = max\{cardinality(children(m)) \text{ where } m \in level(i-1)\}$$

The length of the surrogates can be chosen sufficiently large in advance and although the fixed length leads to static boundaries of the surrogates this can be neglected. Extending every surrogate by one bit doubles the number of children that can be addressed at every level.

Besides the compact, prefix-free and order preserving representation it is also possible to evaluate path expressions on compound surrogates. Evaluating simple path expressions

(i.e., full qualified paths) in MHC is equivalent to point queries on the compound surrogates while other path expressions are equivalent to range queries or combinations of point and range queries. It can be shown that all 13 location steps which are defined in XPath 1.0 and XPath 2.0 can be implemented as simple expressions on compound surrogates.

4.2 The Schema

The implementation of our mapping scheme is based on two relations. The core is a table with three attributes, which we refer to as **xmltriple**. The table **xmltriple** holds the attributes did, val, and surr (see Table 1 for an example with two tuples and 4 bits per surrogate). For each value in a document, **xmltriple** stores the corresponding path information as a compound surrogate and the document, which contains this value, as a document id.

For mapping XML document paths to compound surrogates we use an additional table **typedim** with the two attributes path and surr (Table 2). The table does not contain any information about paths on a per document basis but only stores complete paths to leafs. This reduces the size of **typedim** significantly, so **typedim** is typically very small compared to **xmltriple**.

did	val	surr
1	Rudolf	0010 0001 0001 0001 0000 0000 0000 0000
1	Bayer	0010 0001 0010 0001 0000 0000 0000 0000

Table 1: Relation `xmltriple`

surr	path
0010 0001 0001 0001 0000 0000 0000 0000	/Author[1]/FN[1]
0010 0001 0010 0001 0000 0000 0000 0000	/Author[1]/LN[1]

Table 2: Relation `typedim`

For our measurements we compared four different indexing methods for our proposed mapping scheme. Since we claim that XML indexing is a multidimensional problem we chose the UB-Tree (threedimensional) and three variants of B-Tree compound indexes for the **xmltriple** relation.

4.3 The Query-Rewriting

As mentioned already in Section 1 XML Queries on XML documents have to be rewritten to SQL so that RDBMS are able to process them. The method for our mapping is shown

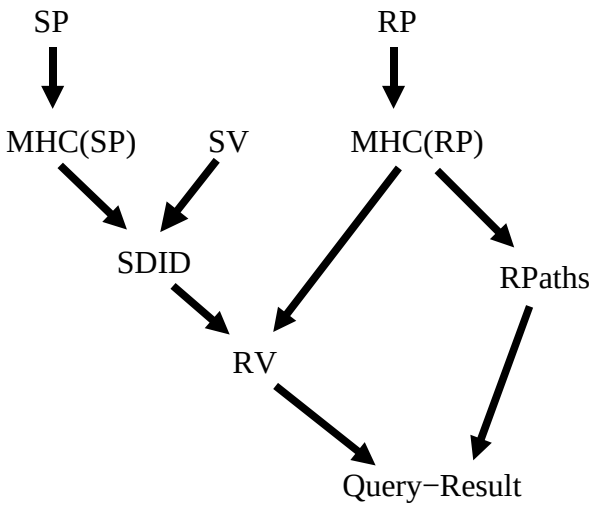


Figure 3: Steps to rewrite the query into SQL

in Figure 3.

To illustrate the rewriting process we have chosen an example query from a typical library scenario. The query returns the title and keywords of all scientifi c papers that were written by a certain author. The same query will later be used in our measurements.

Example 3

```
select titel, keywords
from xmldata
where /author[FN = "Michael" and LN = "Bauer"]
```

The initial query is seperated into the already mentioned two components *selection* and *projection*. The selection consists of a selection path (SP) and selection value (SV), which can be seen in the left branches of the diagram in Figure 3. The paths are mapped to compound surrogates by a lookup in the **typedim** table (MHC(SP)). The compound surrogates and search values are then used to identify the document ids which satisfy the selection predicate (SDID) by querying the **xmltriple** relation.

The projection, which outputs the result values (RV), is processed similarly. The result paths (RP) are transformed into compound surrogates. In the next step the result values are returned by using the retrieved document ids and the compound surrogates as input. More complex queries might require recursive application of this schema.

As we are examining a relational mapping of XML data the queries of each step have to be rewritten to SQL statements. Some of the statements depend on the output of previous queries. This may lead to long statements (especially for many hits in the document ids) and for ease of presentation we only give very short example statements. It would of course be possible to rewrite the query into one large SQL statement, but this would hide

the diversity of the query parts and signifi cant facts about the nature of XML queries.

Example 4 *1. Step: MHC(SP)*

```
select surr from typedim
    where attr like 'Author[%]/FN[1]' order by surr;
select surr from typedim
    where attr like 'Author[%]/LN[1]' order by surr;
```

2. Step: SDID

```
select distinct x1.did from xmltriple x1, xmltriple x2
    where
    (x1.surr = 0b0010000100010001000000000000000
    or x1.surr = 0b00100001000010001000000000000000
    ..............    result of MHC(SP)
    or x1.surr = 0b00101000000100010000000000000000)
    and
    x1.val = 'Michael'
    and
    (x2.surr = 0b0010000100100001000000000000000
    or x2.surr = 0b00100001000010000100000000000000
    or ..............   result of MHC(SP)
    or x2.surr = 0b00101000000100001000000000000000)
    and
    x2.val = 'Bauer'
    and
    x1.did = x2.did
    order by x1.did
```

3. Step: MHC(RP)

```
select surr from typedim
    where attr like 'Title[1]'
       or attr like 'Keyword[%]' order by surr;
```

4. Step: RV

```
select did,val,attr,typedim.surr from xmltriple,typedim
    where
    (typedim.surr = 0b000100010000000000000000000000000
    or
    typedim.surr = 0b001100010000000000000000000000000
    or ..........  result of MHC(RP)
    or
    typedim.surr = 0b00111000000000000000000000000000)
```

```
and ( did = 76 or .....  result of SDid
or did = 9783 )
and typedim.surr = xmltriple.surr
order by did,typedim.surr
```

5 Measurements

5.1 The Data and Measurement Environment

We have chosen a DTD from a digital library scenario for our measurements. The DTD of the documents is a typical specification for scientific papers. We generated 10000 different documents using the XMLGenerator tool [XML]. The raw size of the XML data is approx. 50 MB. We have used a similar distribution [CSF+01] as in the DBLP database [DBL] for authors which results in approx. 400 different authors for 10000 documents. From these documents we generated flat files for bulk loading the databases. The sizes of the database are approx. 25 MB and they differ slightly depending on the used index. For our measurements we ran the already above mentioned query on the different indexed tables.

All measurements were performed with the relational database system TransBase[1], which won the European Information Technology Prize 2001 for its pioneering implementation of UB-Tree indexes. We chose a page size of 2KB and limited the database cache to 128 KB. With a cache size this small not many pages can be kept in the cache. Completely eliminating the cache would severely decrease performance as even index pages would no longer reside in the cache.

The database system was installed on a Sun Ultra 10 (400MHz, 512MB main memory) and the measurements were performed on a Seagate ST39111A (73.4GB) U160 hard disk.

In the following we use abbreviations to denote the different indexing methods.

- *UB-Tree* denotes the indexing with the threedimensional UB-Tree, the index attributes are *did, surr, value*.
- *DidSurr* denotes indexing with a compound B-tree with the index attributes *did* and *surr* (in this order),
- *DidSurr_Val* adds an additional secondary B-Tree index on *val*.
- *SurrValDid* denotes the use of the compound B-tree with the index attributes *surr*, *val*, and *did*.

In addition, we measured two variants of the Edge mapping approach. Edge mapping is explained and discussed in detail in Section 5.4.

We rewrote the query from Section 4.3 to SQL as shown above and measured both, the elapsed time and the number of pages that were accessed for answering the SQL queries.

[1] http://www.transaction.de

The number of retrieved pages is further divided into the overall number of physical page accesses, logical accesses to the index pages, and logical accesses to the data pages. The physical page accesses occur whenever the database system requests a page from secondary storage, i.e. the page is not available from database cache. Logical page accesses occur whenever a page is accessed by the database system. One physical page access leads to at least one logical page access. Several logical page accesses occur if the page is accessed several times, e.g., if tuples on pages are repeatedly read in different stages of query processing. In these cases no physical access occurs if the page is available in the database cache.

We analyze selection and projection separately and for our example data set (10000 documents) the following numbers of tuples were returned for each processing step (Table 3).

Query Step	Selection (SDID)	Projection (RV)
Number of Tuples	104	554

Table 3: Number of tuples returned for each processing step

5.2 Selection

As noted above, we have separated the queries into a selection and projection part. The selection restricts values and compound surrogates and outputs a set of document ids. The set of document ids is then further processed in the projection.

The selection query is graphically illustrated in Figure 4 (due to reasons of visibility the figure only shows 8 point restrictions). The query cuts through the cube as the two dimensions are restricted, while the third is variable. The query is located on two parallel planes orthogonal to the value dimension and is processed using 16 point restrictions (one for each surrogate and value restriction). The compound surrogates are dense in their dimension.

5.2.1 UB-Tree

For the UB-Tree the results of this query are located on the same page with a high propability due to the space-filling Z-curve. As the query is processed by repeatingly stabbing through the threedimensional space pages are read severaly times; caching can be used in this situation to speed up query processing. In this case the caching is intra-query as the query processing itself benefits from the reuse of pages that were already read from secondary storage at earlier stages of query processing.

The consequence is that the average time per page sharply decreases (0.3 ms/physical page), as the page is processed only in memory. This is shown by the measurements in Table 4 and Table 5.

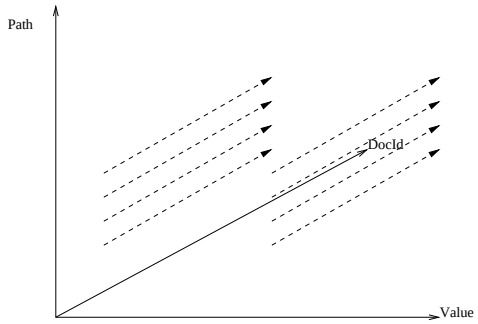


Figure 4: Point restrictions for selection query

Index	log. Idxp.	log. Datp.	phys. Pages	DB size (pages)
UB-Tree	918	782	693	20428
DidSurr	213	11946	11957	14426
SurrValDid	39	22	36	12132
DidSurr_Val	215	705	1208	18656
Edge_Compound	386566	241118	49776	41585
Edge_Secondary	5689	7473	6526	93999

Table 4: Page numbers for the selection

5.2.2 Compound B-Tree *DidSurr*

For a compound B-Tree on the attributes *did* and *surr* the scenario is completely different compared to the UB-Tree. The query processor of the database system cannot use any restriction on the *did* attribute, so it has to start with the smallest value for the compound surrogate dimension, starts to read all data pages and performs a post-filtering. Table 4 shows that the query reads almost all pages of the database. The query is slowest among all the others (Table 5), still it achieves a high page rate (1.7 ms/physical page). This hints that the query is highly supported from caching. This time it is not intra-query caching as with the UB-Tree, but caching from the operating system.

5.2.3 Compound B-Tree *SurrValDid*

In our third measurement the selection is performed with **xmltriple** indexed with a compound B-Tree on the compound surrogate attribute, the value attribute, and the document id attribute. This index exactly supports the restricitions of the query. The number of physical pages read is only 5% of the physical pages read for the UB-Tree (Table 4).

Index	Selection	Projection	Total
UB-Tree	0.26s	8.73s	8.99s
DidSurr	20.3s	1.96s	22.3s
SurrValDid	0.02s	9.18s	9.20s
DidSurr_Val	0.95s	2.16s	3.11s
Edge_Compound	60.3s	0.58s	60.9s
Edge_Secondary	6.65s	2.50s	9.15s
Tupel	104 (DocIds)	554 (Tags)	

Table 5: Running times for selection and projection queries

5.2.4 Compound B-Tree *DidSurr* with secondary index *Val*

A closer look on the restrictions of the selection query and on the result for the three indexes discussed above, show that it is important for the performance of the index to directly support a restriction on the value attribute. Depending on the data, especially if value restricts stronger than structure (the structure here is represented by the surr attribute), a secondary index on the value attribute should improve the scenario for the *DidSurr* index. The creation of the secondary index results in an increase of the database size by approx. 30% but is still below the size of the UB-Tree index. Our measurements show that both, the elapsed time and the number of retrieved pages are reduced as expected (Table 5 and Table 4) although the performance of the *SurrValDid* index is of course not reached.

5.3 Projection

The projection query outputs values and restricts the document ids and the compound surrogates. The restrictions are no range restrictions but point restrictions. The cardinality of the set of points in the document id dimension depends on the result set of the selection query. It is important to note that the result set of the selection query is usually not a range (a range would be quite unlikely). The compound surrogates in the other dimension are restricted to 9 point values (one compound surrogate for title and 8 compound surrogates for keywords). All values which answer the query are consequently located on $9 \times 104 = 936$ parallel straight lines intersecting the three dimensional space. Figure 5 sketches the scenario with three lines due to the sake of clarity.

5.3.1 UB-Tree

Since the straight lines completely stab through the universe (there is no restriction in the value dimension) we can deduce some more information about the processed data by calculating the expected number of page accesses. According to Table 6 the size of the database is 20428 pages. This results in approx. $2^{15} = 2^{3*5}$ pages. This leads to approx. $2^5 = 32$ pages in each of the three dimensions meaning that each of the 936 straight lines

touches 32 pages. This sums up to 29952 logical data page accesses.

Figure 6 presents the page numbers for the projection query from our performance tests. The accessed logical data pages are close to the results calculated above. They are not exactly the same because we have assumed a uniform distribution of the data. This is not the case for the data we used.

Another drawback for the UB-Tree in this measurement is the very high number of physical accesses to the pages. The numbers show that for our simple example query almost one third of the database is being read (6441 pages of 20428 pages) although the result set is very small. Most of these physical page accesses are data pages as only 1% of the overall pages in the database are index pages. The high number of physical page acccesses is a combination of the way the query is processed (as explained above), the distribution of the selection results which are not ranges but spread over the *docid* dimension and the clustering of the UB-Tree which does not favour one or two attributes but treats all attributes in an equal manner.

Index	log. Idxp.	log. Datp.	phys. Pages	DB size
UB-Tree	39700	31043	6441	20428
DidSurr	384	436	305	14426
SurrValDid	6	463	468	12132
DidSurr_Val	384	436	305	18656
Edge_Compound	374	517	508	41585
Edge_Secondary	2278	2488	1353	93999

Table 6: Page numbers for the projection

5.3.2 Compound B-Tree *DidSurr*

The compound index on the attributes *did* and *surr* is the fastest index for projection as the query restricts exactly the index attributes to points. As there are more dids than compound surrogates there are more index pages read than for the *SurrValDid* index.

5.3.3 Compound B-Tree *SurrValDid*

Indexing the **xmltriple** relation with a compound B-Tree (index attributes *surr, value, did*) leads to low page numbers in comparison to the UB-Tree. In constrast the running time of the query is almost the same as for the UB-Tree. A closer look on the way the query is processed reveals that both numbers are reasonable. The projection query restricts on the surrogates and on *did* and there is no restriction on the values. The system starts with reading the data pages starting with the smallest value for value and *did* until it terminates when the surrogate range is processed for the keyword surrogates. The system accesses the B-Tree a second time for the title surrogat. The results of the projection query are determined by post filtering the retrieved pages. The retrieved 463 data pages carry approx. $463 * 100 = 46300$ tuples. The post filtering has to be done for each of the 104 tuples that

resulted from the selection. This leads to $104 * 46300 = 4815200$ overall comparisons. The observation shows that the projection for *SurrValDid* is not I/O bound but CPU bound.

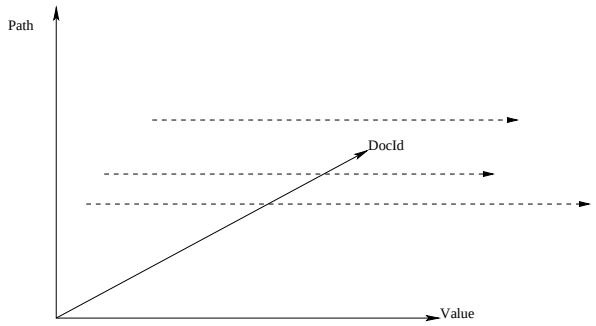


Figure 5: Point restriction for projection query

5.3.4 Compound B-Tree *DidSurr* with secondary index *Val*

In the projection part of the query there is no restriction on the value attribute and consequently the secondary index can not be used by the query optimizer. The resulting measurements are therefore the same as for the *DidSurr* index.

5.4 Comparison with Edge Mapping

A common way to estimate the performance of a mapping scheme is a comparison with the Edge mapping for XML data[FK99]. We use the slightly modified version of the Edge mapping from [CSF+01]. The mapping consists of two tables *roots(id, label)* and *edge (parentid, childid, label)*. The *roots* table contains a tuple for every document with an *id* which identifies the document (document identifier) and *label* for the root tag. The *edge* table contains a tuple for every nesting relationship. Non leaf nodes contain the id of the *parent* node in the parent attribute and the id of the child node in the *childid* attribute. The *label* contains the tag. Tuples for leaf nodes (data elements) carry a NULL value in the *childid* attribute and the data value in the *label* attribute.

Example 5 *The fragment*

```
<paper><title>Fast Query Processing</title></paper>
```

is stored with the tupel (0, paper) in the roots table and with the tuples (0,1, title) and (1, NULL, 'Fast Query Processing') in the edge table.

For our measurements we indexed the root table with a primary compound B-Tree on the attributes (*parentid, childid*) (*Edge_Compound* in Tables 4-6). In a second variant we cre-

ated secondary indexes on each of the attributes *parentid*, *childid*, and *label* (*Edge_Secondary* in Tables 4-6). The second variant comes closest to the measurements in [CSF+01].

As for the previous mapping we also examine selection and projection separatly. Since Edge mapping explicitly stores the graph structure the space requirements for the *edge* table are much higher than for our proposed XML mapping in which we only store the paths from the root to leafs (Table 4).

Following paths in the *edge* table leads to a long series of self joins (depending on the length of the path) in the rewritten SQL queries.

In the first variant the selection is very slow since no restrictions can be used on any index attributes (60.2s) and the RDBMS performs a full table scan. The scenario is different for the projection. Here the Edge mapping is the fastest among our measurements. The restriction on the 104 document ids for the projection leads to a strong restriction on the *parentid* attribute and for every self join the intermediate results are further reduced. In addition the projection query is heavily supported by intra-query caching effects as intermediate results that were already processed for the self-joins can be reused for later stages of the query.

The second variant (which uses secondary indexes on *parentid*, *childid* and *label*) reaches a remarkable database size of 93999 pages (almost half of this size is occupied by the secondary indexes). This is almost 7 times the size of the smallest database size for our proposed multidimensional mapping and is larger than the original raw XML data. Using a hand-tuned query plan we could lower the selection query to 6.65s. We hand-tuned the plan as the original plan used non-optimal join sequences which lead to an original running time of $>$300s. The projection query is also hand-tuned (otherwise $>$145s). It is now among the fastest query as not many self-joins have to be performed to reassemble the paths (we only query for title and keywords which are toplevel tags). In addition the restrictions on surrogates and document ids are also very well supported by the secondary indexes, but the overall running time of selection and projection (9.15s) are severly declined by the tremendous database size.

6 Related Work

Storing, indexing and retrieving XML in database systems got a lot of attention in research over the last years. Among the mapping schemes especially the work of Florescu and Kossmann[FK99] was very influential. Their approach to explicitly store the graph of the XML documents in relations (the Edge mapping) became one of the benchmarks that almost all other mappings (including ours) have to compete with. Besides other static mapping approaches (e.g. [STZ+99]) there were efforts to extract the schema from the data itself using data mining algorithms (e.g. as in the STORED project[DFS99]). The schema is then used to store the data in a relational database. Some proposals for storage though neglect the order of paths in XML documents. Order was only recently discussed in detail[TVB+02]. The proposed Dewey order is very similar to our MHC technique and is originally used for the classification of library items.

Most index structures for XML and semistructured data concentrate on the efficient enconding of paths. Cooper, et al.[CSF+01] present a solution based on fast text enconding using patricia tries (which they evolved into a balanced index structure for secondary storage). Besides other research work (T-index [MS99]) there are also commercial systems available which claim to store XML data "natively" but omit a detailed description [Tam, XIS]. Widom, et al. have published significant work on systems storing semistructured data, XML and query languages [MAG+97, AQM+97]. The Lore system which is based on the OEM model uses different indexes on values and paths to speed up query processing [MWA+98]. We use a similar approach for the indexes of our relational XML mapping.

Recently a technique to speed up XPath location steps using a numbering scheme based on pre and post order of the XML document graph was published and benchmarked using R-Trees and compound B-Trees [Gru02]. In contrast to our MHC proposal which is onedimensional this approach is multidimensional.

7 Summary

We have described a multidimensional mapping scheme for XML and relational database systems. In addition we have analyzed four different multidimensional indexing methods for our mapping (the UB-Tree and three variants of a compound B-Trees) and compared them to the well-known edge mapping. In the overall running times of the queries we were faster in every case than edge mapping and had a up seven times smaller database size. Additionally we have described the orthogonal nature of typical XML queries.

Due to the special orthogonal requirements for selection and projection currently a combination of the two compound B-Trees seems to be the most promising one with respect to elapsed query time. The use of two indexes obviously requires more maintenance work for insertion as a tuple is effectivly inserted two times into the indexes. Another drawback is the consumption of additional storage for the second index (the use of additional indexes renders one variant of the edge mapping unusable). A very promising result is the use of a secondary index in our mapping to avoid the limitations of one index. This combines the advantages of two indexes while reducing space requirements.

The research on the multidimensional model, the orthogonal requirements for selection and projection, and the importance of choice of indexes has shown that typical XML queries carry restrictions on values and structure. Depending on data and queries the performance of indexes may strongly vary with the selectivity of value and structure attributes. This has to be especially taken care of when chosing indexes in database schemas for storing XML.

Acknowledgements

Part of this work is funded by DFG (German Research Foundation) within the research initiative “V3D2” (“Distributed Processing and Delivery of Digital Documents”).

References

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel Query Language for Semistructured Data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[CSF+01] Brian Cooper, Neal Sample, Michael J. Franklin, Gisli R. Hjaltason, and Moshe Shadmon. A Fast Index for Semistructured Data. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 341–350, 2001.

[DBL] DBLP, Uni Trier, http://dblp.uni-trier.de/.

[DFS99] Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing Semistructured Data with STORED. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 431–442. ACM Press, 1999.

[FK99] Daniela Florescu and Donald Kossmann. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Rapport de Recherche No. 3680, INRIA, Rocquencourt, France, May 1999.

[Gru02] Thorsten Grust. Accelerating XPath Location Steps. In *SIGMOD 2002, Proceedings ACM SIGMOD International Conference on Management of Data, June, 2002, Madison, Wisconsin, USA*. ACM Press, 2002.

[MAG+97] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, and Jennifer Widom. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):54–66, 1997.

[MRB99] Volker Markl, Frank Ramsak, and Rudolf Bayer. Improving OLAP Performance by Multidimensional Hierarchical Clustering. In *Proc. of IDEAS Conf., Montreal, Canada*, 1999.

[MS99] Tova Milo and Dan Suciu. Index Structures for Path Expressions. In *Database Theory - ICDT ’99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*, pages 277–295. Springer, 1999.

[MWA+98] Jason McHugh, Jennifer Widom, Serge Abiteboul, Qingshan Luo, and Anand Rajaraman. Indexing Semistructured Data. Technical report, February 1998.

[STZ+99] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *VLDB’99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 302–314. Morgan Kaufmann, 1999.

[Tam] Tamino XML Server, SoftwareAG, http://www.softwareag.com/.

[TVB+02] Igor Tatarinov, Stratis Viglas, Kevin S. Beyer, Jayavel Shanmugasundaram, Eugene J. Shekita, and Chun Zhang. Storing and Querying Ordered XML Using a Relational Database System. In *SIGMOD 2002, Proceedings ACM SIGMOD International Conference on Management of Data, June, 2002, Madison, Wisconsin, USA*. ACM Press, 2002.

[XIS] XIS XML database, Excelon, http://www.exceloncorp.com/.

[XML] XML Generator, IBM Alphaworks, http://www.alphaworks.ibm.com.

OraGiST - How to Make User-Defined Indexing Become Usable and Useful

Carsten Kleiner, Udo W. Lipeck
Universität Hannover
Institut für Informationssysteme
FG Datenbanksysteme
Welfengarten 1
30167 Hannover
{ck | ul}@dbs.uni-hannover.de

Abstract: In this article we present a concept for simplification of user-defined indexing for user-defined data types in object-relational database systems. The concept is based on a detailed analysis of user-defined indexing in ORDBS on one hand, and features of generalized search trees (GiST) as an extensible indexing framework on the other hand. It defines a minimal interface to be implemented in order to use GiST within ORDBS; this greatly simplifies the process of implementing user-defined indexes. The effectiveness of the approach is illustrated by performance experiments carried out on a prototypical implementation of our concept. For the experiments we have used new specialized spatial data types, that store spatial as well as thematic information within a single attribute. These data types facilitate advanced spatial analysis operators. The experiments show great performance improvements on these operators by using multidimensional user-defined index structures based on R-trees when compared to system-provided indexes.

Keywords: object-relational databases, user-defined indexing, data cartridge, user-defined datatype, advanced spatial analysis operators

1 Introduction

1.1 Motivation

In recent years object-relational database systems (ORDBS) have advanced past research prototypes and are becoming commercially available. One of the key advantages over traditional relational databases is the possibility to use user-defined datatypes (UDTs). It is a very important feature in many advanced applications, because adequate modeling of non-standard domains requires such types. This can be inferred from the increased popularity of object-oriented modeling which implicitly facilitates the use of arbitrary datatypes.

Also in the spirit of object-oriented modeling these new UDTs will have specific functions operating on them, usually called methods in object-oriented technology. The most impor-

tant of these methods from a database perspective are methods that can be used in queries to select objects from tables or join objects from different tables. These methods are also called *operators* in object-relational terminology. In commercial settings these new features will only be used, if they achieve a performance comparable to standard datatypes and functions in traditional relational databases.

Since UDTs can be defined arbitrarily by the user, it is impossible for the ORDBS vendor to provide efficient support of all such operators in selection and/or join queries. Consequently they provide *extensible* indexing and query optimization features. The author of the UDT has to implement these features using domain-specific knowledge. The complete package of UDT, operators, indexing and query optimization is also called a *cartridge* (or, depending on the particular vendor, *extender* or *data blade*), since it can be plugged into the database server similar to cartridges in hardware. The enhanced database server then efficiently supports UDTs transparent to the end-user.

A big obstacle, however, is that the implementation of user-defined indexing is pretty complex and time-consuming. Moreover as it is, it has to be carried out completely from scratch for every UDT that requires efficient query support. In order to overcome this and make user-defined indexing more usable, a *generic* indexing framework such as generalized search trees (GiST; [HNP95]) should be used. This has the advantage of already providing most of the required functionality for several different index structures. These structures can easily be specialized to obtain a specific index for a particular datatype.

Therefore it would be desirable to have an easy to use tool that combines a particular UDT from an ORDBS with a concrete extension of an index framework automatically and leaves only the (very few as will be shown in this article) type and operator specific implementation parts to the user. In this article we will first investigate what the type and operator specific details of such a definition under some reasonable assumptions are. The result is a concept for greatly simplifying the usage of an indexing framework in ORDBS. We will then present the design of a prototypical implementation of such a tool, called *OraGiST*, which facilitates the use of GiST as user-defined index structures in the ORDBS Oracle 9*i*. The tool which was developed in our group does as much work as possible automatically, leaving the programmer with just very few tasks[1] to be solved, before being able to use GiST inside Oracle. While OraGiST is specifically designed for Oracle, similar extensions for other ORDBS based on the previous concept would also be possible.

The effectiveness of this approach is illustrated by some sample results from spatial data. In advanced applications one would like to combine spatial selection criteria with other thematic attributes in a single operator. Queries such as *select all cities in a given query window where the population density is higher than a given value and the percentage of male inhabitants is more than 55 percent* could be answered efficiently by such complex operators. We will show that using high-dimensional R-tree-like GiST extensions developed by using OraGiST is more efficient than working with classical indexes. It is even more efficient than using the DBMS provided spatial indexes on the spatial component.

[1] If using one of the predefined GiST extensions it takes in the order of minutes to implement the required parts.

1.2 Related Work

The concept of object-relational database systems (ORDBS) has been described in [SM96] and [SB99]. Most commercial DBMS that were relational can be called object-relational to a certain degree today. Probably the most commonly used systems are Oracle 9*i* and IBM DB2. In the open-source area e. g. PostgreSQL is also object-relational; it is based on the oldest ORDBS Postgres (cf. [SR86]). Requirements specific to user-defined indexing in ORDBS are explained in [SB99].

A good overview of advanced spatial applications can be found in [RSV01]. In particular indexing in spatial databases is reviewed in [GG98]. RSS-Trees were introduced in detail in [KL00]. The introduction of user-defined types in ORDBS requires easily extensible indexes in order to facilitate efficient querying of UDTs. Therefore generic index structures such as generalized search trees have to be used. A similar idea to ours has been described in [RKS99] for the CONCERT DBS. It allows indexing based on concepts but is more focused on spatio-temporal data and is restricted in the indexes to be used since it requires a hierarchical partitioning of space. The basic insight that only very few features are actually specific to a data type in indexing is also used in our approach. More advanced research on query optimization for user-defined types is described in [Hel98]. Another implementation for advanced index structures was presented in [BDS00]. It is written in Java and focuses on main memory index structures. When Java has become more efficient and systems even more powerful in terms of main memory this will probably also be a very interesting approach. The combination of an extensible indexing framework with a commercial ORDBS has to the best of our knowledge never been researched before. Also the idea of using higher-dimensional indexing for advanced spatial analysis has never been analyzed in such detail. Finally no experimental results on the efficiency of real user-defined indexing in ORDBS have been reported yet.

2 Current Situation

Generalized search trees (GiST; [HNP95]) are search trees that combine and implement the common features of tree-based access methods such as insertion and searching but still allow to adjust the classical operations to a particular data type and indexing structure. This is achieved by an implementation using certain extensible methods in the general algorithms that have to be implemented by the user for the particular data type and indexing structure by object-oriented specialization later; these methods are consistent, penalty, pickSplit, union and optionally compress and decompress. Due to space constraints details on GiST are omitted; they can be found in the full version of this paper ([KL02]).

A file-based implementation of generalized search trees called `libgist` has been made available under `http://gist.cs.berkeley.edu`. It is written in C++ and provides the general tree along with several extensions for most of the trees proposed in the literature such as B-Tree, R*-Tree and SS-Tree. Its completion was reported in [Kor99]. Since it operates on index files, it cannot be used directly in conjunction with a commercial ORDBS, where index files should be under control of the DBS as well, e. g. for transactional

correctness reasons. Moreover the interfaces need to be linked together in order to use a GiST specialization as an index inside the ORDBS. In addition ORDBS users want to use the given and other GiST-based index structures for their own user-defined types. This requires a mapping of database types and operators to GiST objects and predicates.

In order to implement **extensible user-defined indexing** - for example in Oracle 9*i* - certain methods have to be programmed: IndexCreate, IndexInsert, IndexDrop, IndexStart, IndexFetch, IndexClose, IndexAlter and IndexUpdate. These are later automatically invoked by the database server when executing regular SQL commands using the UDTs. Again due to space constraints details are omitted here, but can be found in the full version of the paper.

User-defined data types in ORDBS need data type specific **query optimization** to be able to use them efficiently. In Oracle this is implemented by providing special interfaces for extensible optimization. Functions defined in these interfaces are called automatically by the database server, if an implementation of the interface for the particular data type is registered with the server. This way extensible optimization is as directly integrated into the database server as possible for arbitrary data types. The interfaces of the extensible optimizer are described in detail in system documentation.

3 Concept of OraGiST

In order to connect the file-based implementation of generalized search trees with the indexing interface of the ORDBS we have firstly analyzed which components of the mapping between a generic index and an ORDBS are generic themselves and which are dependent on the particular data type in question. Consequently, the implementation of a connector component between ORDBS and generic index consists of two main parts. We have called our prototypical implementation `OraGiST`, since it operates based on `libgist` and Oracle 9*i*. An overview of the architecture of `OraGiST` is given in figure 1.

As explained before, the tool `OraGiST` mainly consists of two components. The first component (called OraGiST library) is independent of the particular data type, user and GiST extension. It provides functionality for calling the appropriate generic index methods inside functions of the ORDBS extensible indexing interface (cf. section 2) on the one hand. Also it facilitates storing of index information of a file-based tree index structure inside an ORDBS on the other hand. These are the upper two associations in figure 1.

The second component (OraGiST toolbox in figure 1) is dependent on the data type and thus index structure specialization. Consequently, it cannot work completely without programmer interaction. It can rather be a support tool providing the user with method prototypes, and taking over all generic tasks of this part of the user-defined index development process. In particular, only four methods have to be implemented for the particular data type by the database programmer. Firstly, it has to be defined which GiST specialization is to be used for a particular ORDBS user-defined data type[2]. This is done by implement-

[2]This specialization has to be written separately in advance; it is not a big restriction, though, since many important index structures for different domains are already provided with `libgist`. Also because of the

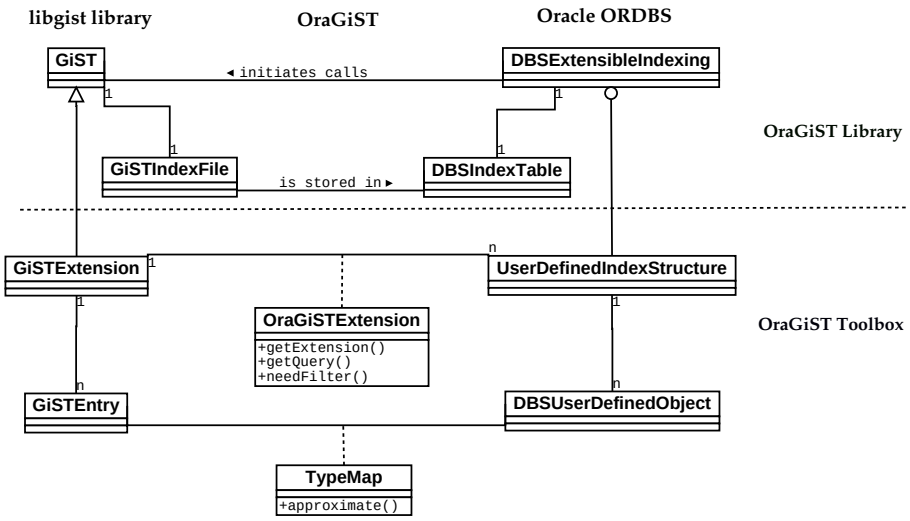


Figure 1: Architecture and Functionality of `OraGiST`

ing the method `getExtension`. Secondly, the mapping of database operators to index structure methods implementing the desired predicate has to be declared by implementing the method `getQuery` by the developer. The implementation of these two methods typically consist of only a single line returning a pointer to the particular object required.

Also the mapping between ORDBS data type and index structure entry has to be implemented. Since often not the exact objects but rather approximations are inserted into an index, this method is called `approximate` (cf. MBRs in spatial indexing). More generally speaking this method implements the compress method of the original GiST concept. This method is the only one that may incur considerable programming work, since it must implement the mapping of a database object to an index object; in the case of spatial data in Oracle 9*i*, for instance, the rather complex definition of the database geometry type has to be investigated in C to compute the MBR of the object.

Finally, results from an index lookup may not be results to the original query, if the index entries are approximations of the original objects. For spatial data e. g. not all objects whose MBR intersect a given object do intersect the object themselves; thus usually the so-called filter-and-refine strategy of query execution is used for spatial queries. This is reflected in `OraGiST` by means of a method `needFilter`. It has to return `true`, if results from the index scan still have to be tested, before the query result is determined, and `false` else. For the programmer this is only one additional line of code in the current version of `OraGiST`, since as testing method the functional implementation of the operator which is required by the ORDBS anyway may be used. If such testing is required, after registering the index with `OraGiST`, the toolbox will automatically take care, that results of an index scan are additionally filtered by the functional implementation of the operator

framework character of `libgist` such extensions are supported very well.

```
CREATE TYPE twoIntegerGeometry AS OBJECT (
   geom OGCGeometry,
   theme1 INTEGER,
   theme2 INTEGER,
   -- type-specific methods);
```

Figure 2: Data type definition for combining spatial and thematic information

in the ORDBS.

As stated earlier based on this concept, we have developed a prototypical implementation of a connector between GiST implementation `libgist` and Oracle 9*i* as ORDBS. This simplifies user-defined indexing to such a degree that it is really usable with acceptable implementation effort. If one of the predefined GiST extensions is to be used as index only the small `OraGiST` toolbox classes have to be implemented. But even if a new GiST extension is required as index development time is greatly reduced. This is due to the extensibility features of GiST which facilitate simple definition of new index structures as long as they can be expressed in the GiST framework. We will illustrate that user-defined indexing is also very useful for UDTs by presenting performance experiments of using such indexes for advanced spatial analysis operators in the next section.

4 Case Study: Advanced Spatial Analysis

4.1 Data Types and Operators

Spatial index structures have been researched in great detail in the past. Operators supported by most of these indexes are spatial selections such as overlap or window queries. Some also support spatial join or nearest neighbor queries. Common for all these queries is that they only use the spatial information of objects for determining the query results.

Recent spatial applications on the other hand tend to use queries that combine spatial and thematical information in finding the desired objects. One could e. g. be interested in all counties in a given window where the median rent is below a given value. Or using even more thematic information, one could ask for all those counties where, in addition, the population is higher than another fixed value. For these queries traditional spatial indexes are only partly helpful, since they only support the spatial selection part of the queries. Especially in cases where the stronger restrictions are on the thematic attributes (e. g. a very low threshold value for median rent) the spatial index does not help at all. Since in many cases it is difficult or impossible to predict which queries will be used on a given table[3], it would be desirable to have a combined index that supports spatial-thematic queries of different selectivities equally well, regardless of the particular query parameters.

[3] Also any type of query could be posed equally often.

Consequently we suggest to define user-defined data types in ORDBS which combine spatial and thematic attributes in a single type[4]. A sample definition for `twoIntegerGeometry` combining a spatial and two numerical thematic attributes is given in figure 2. We use a type `OGCGeometry` as an implementation of the OpenGIS Consortium simple features specification here. Operators on these newly defined types can be conjunctions of spatial operators and operators on the types of the thematic attributes. In the sequel we will investigate the frequently used operator `twoBetweenOverlap` on the aforementioned type, which is a conjunction of spatial overlap operator and between operator on the thematic attributes. Since `twoIntegerGeometry` combines information from orthogonal domains in a single type, it is straightforward to use higher dimensional spatial indexes for the newly introduced operator. In particular, we obtain a four-dimensional domain for `twoIntegerGeometry`.

4.2 Performance Evaluation

An extensive performance evaluation for different datatypes can be found in the full version. If we use one thematic attribute in addition to the spatial attribute and thus use type `integerGeometry` and operator `betweenOverlap`, we obtain the results depicted in figure 3. Indexing options compared in this work are user-defined three-dimensional versions of R*- and RSS-Tree ([KL00]), as well as Oracle spatial indexes on the spatial component of such objects[5]. For the latter indexes the thematic attribute has to be checked separately in order to compute a correct query result. Figure 3 shows that this separate processing incurs a big overhead and thus leads to a pretty bad performance. User-defined indexes perform much better with the R*-tree outperforming the RSS-tree by a small margin. The unstable performance of the predefined indexes is due to them only considering the spatial component. Consequently, when the stronger restriction in the query parameters is on the spatial component they perform better, but worse if the stronger restriction is on the thematic attribute. Figure 3 clearly shows the usefulness of user-defined indexing, since only by using it, an efficient and predictable query performance can be achieved for the user-defined type considered.

Since the built-in indexes already performed bad on `integerGeometry` we do not consider them an alternative[6] for the more complex type `twoIntegerGeometry`. Figure 4 illustrates the performance of different user-defined indexes on queries using the operator `twoBetweenOverlap`. In particular the variants of the user-defined RSS-tree for different dimensionalities are compared; also the four-dimensional R*-tree can be compared with the RSS-tree. The figure clearly shows the benefits of using a multidimensional index structure as compared to a lower dimensional one, since, at least for low to medium dimensional data, the more dimensions are indexed the better the performance is. Also,

[4] Using a user-defined index on the attribute combination without defining a new type was not evaluated due to current technical restrictions that do not allow such a definition. Actually the introduction of such combined types should be subject to some automatic generation based on a specification of the attributes.

[5] Spatial indexes provided with ORDBMS currently do not support more than two dimensions.

[6] In fact, experiments have shown that they perform by far worse than on `integerGeometry`. Therefore we omit these results here.

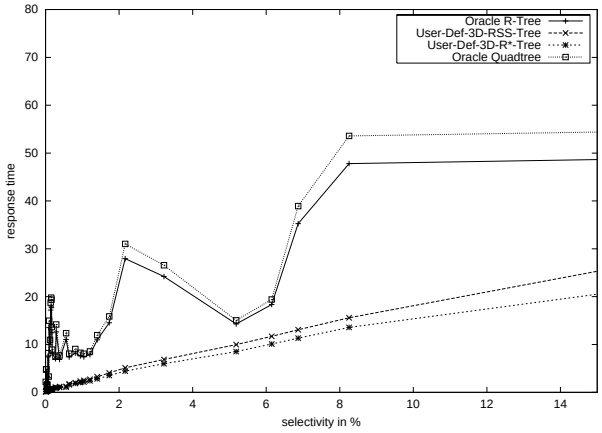


Figure 3: Performance Evaluation on 2D spatial data plus one thematic dimension

different from the indexes on `integerGeometry`, this time the full-dimensional RSS-tree is faster than the R*-tree for small selectivities. Consequently it is difficult to say which of the two variants should be preferred in real applications. But, since both perform almost equally well, the RSS-tree has the advantage of a faster index creation time due to its simpler, namely linear, insertion algorithm. More details can be found in the full version. Nevertheless we can already conclude that the performance gain from using a more appropriate index for a particular data type illustrates, that an easily adaptable indexing framework is required. Only by doing so one can obtain optimal performance with acceptable effort for each of the many possible UDTs.

5 Conclusion and Future Work

5.1 Conclusion

In this article we have presented a concept on how to integrate a flexible extensible indexing framework into recent commercial ORDBS. The need for this integration was motivated by the new features of ORDBS, namely the possibility to use user-defined data types. Since these types also have type-specific operators, type-specific indexes will be required in order to process user-defined operators efficiently.

The concept of integrating generalized search trees into an ORDBS was then applied to specialized spatial data types to prove its feasibility and also its effectiveness. In particular, spatial data enriched with one or two thematic attributes was used as data type. Operators on these types answer queries such as *retrieve all objects that lie in a given area where the median rent is below a given threshold value*. These operators are very frequent in advanced spatial analysis and therefore need index assistance.

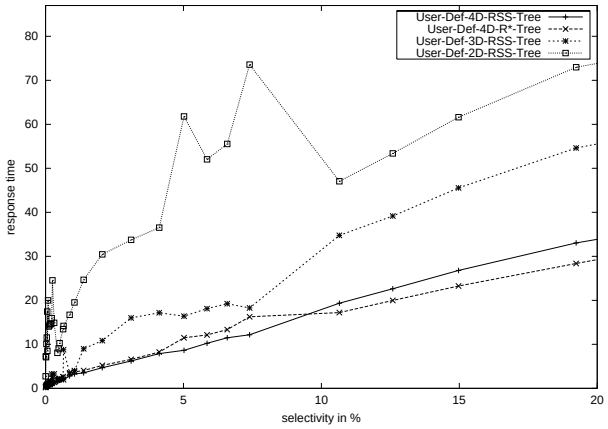


Figure 4: Performance Evaluation on 2D spatial data plus two thematic dimensions (standard query)

Our experiments showed that viewing the objects as elements of a three- (for one thematic attribute) or four-dimensional (for two thematic attributes) space and using spatial indexes extended to three or four dimensions shows best query performance. Overall the experiments showed the huge performance gain achieved by using user-defined indexing as compared to using system-provided indexes. The concept of OraGiST uses object-oriented techniques to simplify the development in two ways: firstly, if an existing index structure is used only the implementation of the interface defined by OraGiST is required. If on the other hand a different index is desired, its development is supported by using an extensible indexing framework such that only a new spezilization has to be developed.

5.2 Future Work

The concept has so far only been implemented in a prototype fashion. Firstly the implementation is currently restricted to GiST and Oracle 9*i*. An extension to other indexing frameworks and more importantly other ORDBS should be evaluated. Also the very good performance results shown previously only hold for rather simple objects. If we consider complex polygons in the spatial component, we currently obtain good results only for very small selectivities. These results could probably become better, if the OraGiST concept is extended to allow for a direct implementation of the refine step during two step query processing in the connector class. Currently the *functional* implementation (fallback) of an operator inside the DBS is used for refinement.

Our concept and prototype implementation should be tested with different, especially more complex, user-defined data types and operators. We expect the results to be even more impressive with such types, since no system-provided indexing is available at all. Also, tests with different operators on the given types, especially different spatial operators, could give an interesting overview of overall performance. Moreover to provide a com-

plete data cartridge fine tuning of user-defined indexing by implementing user-defined cost and selectivity estimation is also missing so far. Conceptually we think, that an extensible framework-like approach could greatly simplify implementation for a particular UDT, similar to user-defined indexing. To be able to use such techniques a parametrized approach to cost and selectivity estimation in ORDBS has to be developed. In particular the processing of data type specific join queries could benefit significantly from optimized query execution, both by indexing and by cost and selectivity estimation. Research on data type specific joins in ORDBS is currently only in its early stages.

Bibliography

[BDS00] J. van den Bercken, J. P. Dittrich, and B. Seeger. javax.XXL: A Prototype for a Library of Query Processing Algorithms. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, May 16-18, 2000*. ACM Press.

[GG98] Volker Gaede and Oliver Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[Hel98] Joseph M. Hellerstein. Optimization Techniques for Queries with Expensive Methods. *ACM Transactions on Database Systems*, 23(2):113–157, June 1998.

[HNP95] J.M. Hellerstein, J.F. Naughton, and A. Pfeffer. Generalized Search Trees for Database Systems. In: *Proceedings of the 21th International Conference on Very Large Data Bases, Zurich, Sept. 11-15, 1995*. Morgan Kaufmann Publishers.

[KL00] Carsten Kleiner and Udo W. Lipeck. Efficient Index Structures for Spatio-Temporal Objects. In: *Eleventh International Workshop on Database and Expert Systems Applications (DEXA 2000; 4-8 September 2000, Greenwich, UK)*. IEEE Computer Society Press.

[KL02] Carsten Kleiner and Udo W. Lipeck. OraGiST - How to Make User-Defined Indexing Become Usable and Useful. Technical Report DBS 01-2002, Institut für Informationssysteme, Universität Hannover, September 2002.

[Kor99] Marcel Kornacker. High-Performance Extensible Indexing. In: *Proceedings of the 25th International Conference on Very Large Data Bases, Edinburgh, Sept 7-10, 1999*. Morgan Kaufmann Publishers.

[RKS99] Lukas Relly, Alexander Kuckelberg, and Hans-Jörg Schek. A Framework of a Generic Index for Spatio-Temporal Data in CONCERT. In: *Spatio-Temporal Database Management – Int. Workshop STDBM'99, Edinburgh, Sept. 10-11, 1999*. Springer-Verlag.

[RSV01] Philippe Rigaux, Michel Scholl, and Agnes Voisard. *Spatial Databases: With Application to GIS*. Morgan Kaufmann Publishers, 2001.

[SB99] Michael Stonebraker and Paul Brown. *Object-Relational DBMSs – Tracking the Next Great Wave (Second Edition)*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, 2^{nd} edition, 1999.

[SM96] Michael Stonebraker and Dorothy Moore. *Object-Relational DBMSs – The Next Great Wave – First Edition*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, 1^{st} edition, 1996.

[SR86] Michael Stonebraker and Lawrence A. Rowe. The Design of POSTGRES. In: *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*. ACM Press.

Effizientes Routing in verteilten skalierbaren Datenstrukturen

Erik Buchmann, Klemens Böhm
{buchmann|kboehm}@iti.cs.uni-magdeburg.de
Arbeitsgruppe *Data and Knowledge Engineering*
Otto-von-Guericke Universität, Magdeburg

Abstract: Verteilte skalierbare Datenstrukturen (SDDS) besitzen große Bedeutung, insbesondere als Grundlage der Realisierung von innovativen Web-Diensten. Die Knoten einer SDDS verwalten (Schlüssel, Wert)-Paare sowie Kontaktinformation über andere Knoten. Diese Kontaktinformationen werden für das Routing von Nachrichten zwischen den SDDS-Knoten benötigt. Dieser Artikel untersucht, wie sich das Caching von Kontaktinformation und die Auswahl der Schlüsselabbildung, d.h. der Abbildung der Datenobjekte auf den Schlüsselraum der SDDS, auf das Routing auswirkt. Unser Hauptergebnis ist die Erkenntnis, dass Caching insbesondere in Verbindung mit einer nachbarschaftserhaltenden Schlüsselabbildung vorteilhaft ist.

1 Einführung

Verteilte skalierbare Datenstrukturen (SDDS) [LNS96, KW94] dienen zur verteilten, selbstorganisierenden Verwaltung von Dictionary-Datentypen, also von Mengen aus (Schlüssel, Wert)-Paaren. Wir sehen viele Anwendungen insbesondere im Web-Kontext, beispielsweise verteiltes Web-Caching [IRD02], Annotations- [RMW94] oder Backlink-Services [CGM99]. Unser neues Projekt *Fairnet* zielt auf die Weiterentwicklung von SDDS ab. Wir erhoffen uns mittelfristig Antworten auf die folgenden Fragen: Wie sieht eine SDDS-Realisierung aus, die Caching, Lastbalancierung, Fehlertoleranz etc. beinhaltet? Wie sehen Self-Tuning Mechanismen in diesem Kontext aus? Was ändert sich, wenn sich einzelne Knoten unkooperativ verhalten? Dieser Artikel beschreibt nun einige Voruntersuchungen, die wir mit einem von uns entwickelten Prototypen durchgeführt haben, um die Leistungseigenschaften von SDDS besser zu verstehen und ihre Performanz zu steigern.[1]

Mit SDDS verwaltet jeder Knoten eine Menge von Schlüsseln. Er leitet eine Operation – z.B. eine Anfrage oder eine Einfüge-Operation – die er nicht selbst bearbeiten kann, an einen aus seiner Sicht geeigneteren Knoten weiter. Jeder Knoten besitzt daher eine *Kontaktliste* in Form einer Menge von Knoten zusammen mit der möglicherweise veralteten Information, welchen Teil des Schlüsselraumes jeder dieser Knoten verwaltet.

[1] Die durchgeführten Experimente sind teilweise vorläufig, da unser Cluster zur Simulation eines großen Network of Workstations noch im Aufbau begriffen ist, wir daher nur mit relativ kleinen Datenmengen experimentieren konnten. Unser experimenteller Aufbau ist aber in Größenordnungen durchaus vergleichbar mit dem in [LNS96, Abe01].

Dieser Artikel untersucht nun, wie einfache Maßnahmen, die die Lokalität in Zugriffsmustern besser berücksichtigen als bisherige SDDS-Implementierungen, zu einer verbesserten Performance führen. Wir konzentrieren uns zwei Aspekte: (1) Auswahl der Knoten in den Kontaktlisten, (2) Einfluß der Abbildung der zu verwaltenden Datenobjekte auf den Raum der möglichen Schlüssel. Unser Kostenmaß ist ebenso wie in [LNS96, KW94] die Anzahl der Message Hops; die Topologie des Netzwerks bleibt außen vor. Hinsichtlich (1) lassen sich existierende SDDS-Implementierungen in zwei Kategorien aufteilen. In der ersten Kategorie (z.B. CAN, P-Grid) ist fest vorgegeben, welche Knoten in der Kontaktliste enthalten sind; das Anfragemuster bleibt unberücksichtigt. Im zweiten Fall (z.B. [KW94]) ist die Länge der Kontaktliste nicht begrenzt. Das ist insbesondere dann unrealistisch, wenn Knoten nur einen kleinen Teil ihrer Ressourcen einbringen wollen. Wir untersuchen, inwieweit existierende Caching-Strategien für die Verwaltung von Kontaktlisten von SDDS geeignet sind, und wir entwickeln eine neue Strategie speziell für diesen Kontext. Es zeigt sich allerdings, dass ausgefeilte Caching-Strategien hier keinen deutlichen Vorteil bringen. Bezüglich (2) haben wir beobachtet, dass bisherige Abbildungen von Datenobjekten in den Schlüsselraum darauf abzielen, den Wertebereich möglichst gleichmäßig auszunutzen. In unserem Kontext ist es dagegen wichtig – und der Artikel wird dies erklären und experimentell nachweisen –, ähnliche Datenobjekte auf ähnliche Schlüssel abzubilden.

2 Content-Adressable Networks

Content-Addressable Networks (CAN) [RFH+01] sind eine Ausprägung von SDDS, bei der jeder Knoten ein multidimensionales Schlüsselintervall verwaltet. Abbildung 1 zeigt beispielhaft den Aufbau eines CAN aus 14 Knoten und die Zuordnung von Ausschnitten des Schlüsselraums zu Knoten. Dies ist unabhängig von der Netzwerktopologie; das CAN etabliert ein virtuelles overlay-Netzwerk. Jeder CAN-Knoten kennt seine Nachbar-Knoten und weiß, welchen Ausschnitt des Raums sie verwalten. In Abbildung 1 hält Knoten 8 beispielsweise Informationen über die Knoten 0, 3, 4, 7 und 9. Die Schlüssel der SDDS sind das Ergebnis der Abbildung von anwendungsspezifischen Datenobjekten auf den SDDS-Schlüsselraum, der *Schlüsselabbildung*. Beispielsweise bildet eine Hash-Funktion URI auf Vektoren von Integer-Werten ab, gegeben den Schlüsselraum $\mathbf{Z}^n$.

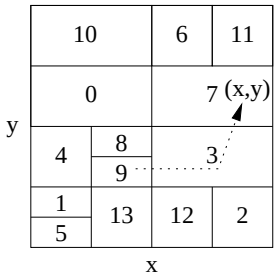


Abbildung 1: Schlüsselraum eines CAN und Beispiel-Anfrage.

Jeder CAN-Knoten kann Operationen wie Einfügen oder Anfragen absetzen. Jede SDDS-Operation spezifiziert dabei einen Zielpunkt im SDDS-Schlüsselraum. Eine Operation löst mindestens eine Nachricht aus, die häufig erst über mehrere Knoten (Hops) zum Ziel gelangt. Das Weiterleiten dieser Nachrichten vom Ausgangsknoten der Operation zum Zielpunkt, der möglicherweise zunächst unbekannt ist, wird allgemein als *Routing* bezeichnet. Abbildung 1 zeigt eine Beispiel-Anfrage des Knotens 9 zum Ziel (x,y) mit zwei Hops.

Caching von Kontaktinformationen in CAN. Im ursprünglichen Entwurf [RFH+01] führt eine Operation in einem d-dimensionalen CAN mit n Knoten zu $O(n^{1/d})$ Message Hops. Bei kleinem d ist diese Zahl zu groß. Bei großem d dagegen ist die Anzahl der zu verwaltenden Nachbarn sehr hoch, die dann bei Änderungen umgehend informiert werden müssen. Um die Anzahl der Nachrichten zu verringern, untersuchen wir im folgenden den Fall, dass die Kontaktliste eines CAN-Knotens nicht nur dessen Nachbarn, sondern zusätzlich weitere Knoten enthält, die der Knoten beim Routing mit einbezieht. Es wird also ein Cache mit limitierter Größe für Kontaktinformation angelegt. Existierende SDDS verfügen zwar über einen Cache, allerdings ist seine Größe entweder nicht begrenzt (vgl. z.B. [KW94]), oder der Cache enthält unabhängig vom Anfragemuster [Abe01] eine fest vorgegebene Auswahl von Knoten. Da beide Fälle nicht optimal sind, wollen wir in diesem Artikel Caching-Strategien und Eigenschaften des Caches untersuchen. Wir orientieren uns dabei an existierenden SDDS-Implementierungen, indem wir veraltete Kontaktinformationen im Cache zulassen, und den Versand zusätzlicher Nachrichten für Aufbau und Verwaltung des Caches vermeiden.

Auswirkungen des Kontaktinformationen-Cachings in CAN. Wir untersuchen in einem ersten Schritt, ob das Caching von Kontaktinfomationen überhaupt sinnvoll ist, und quantifizieren den Einfluß der Cache-Größe auf das Routing-Verhalten. Die Cache-Größe wurde in sechs Schritten von 'keinem Cache' auf 'unbegrenzte Cache-Größe' gesteigert. Als Verdrängungsstrategie wurde Random gewählt. Mit *Random* entscheidet ein gleichverteilter Zufallszahlenstrom über die zu verdrängenden Einträge. Das Zugriffsmuster bleibt unberücksichtigt.

Die Ablaufumgebung ist ein CAN von 1.000 Knoten, in dem 50.000 gleichverteilte Abfragen über gleichverteilte Daten abgesetzt werden. Diese Gleichverteilungen sind der für den Cache ungünstigste Fall, da keine Cache-Einträge bevorzugt nachgefragt werden.

Abbildung 2 visualisiert die Ergebnisse dieses Experiments. Die Darstellung ist durch Durchschnittswertbildung über ein Intervall von jeweils 1.000 Operationen geglättet. Wenig überraschend ist der fallende Grenznutzen der Cache-Größe. Der Graph der unbegrenzten Cache-Größe überschneidet sich bereits vollständig mit dem Graph für 500 Einträge. Ebenfalls zu erwarten war die verlängerte Zeitspanne bis zum Erreichen eines stabilen Zustands bei größeren Caches, da ein kleiner Cache schneller gefüllt ist.

Beachtenswert sind jedoch zwei Aspekte: Zum Einen ist auch ein kleiner Cache bereits sehr wirksam. Schon eine Cache-Größe von 2% der Gesamtzahl der Knoten reduziert die Zahl der Hops um $1/3$. Zum Anderen tritt diese Verbesserung schon nach wenigen ausgeführten Operationen pro Knoten ein. Auch bei einer kurzen Verweildauer der Knoten im

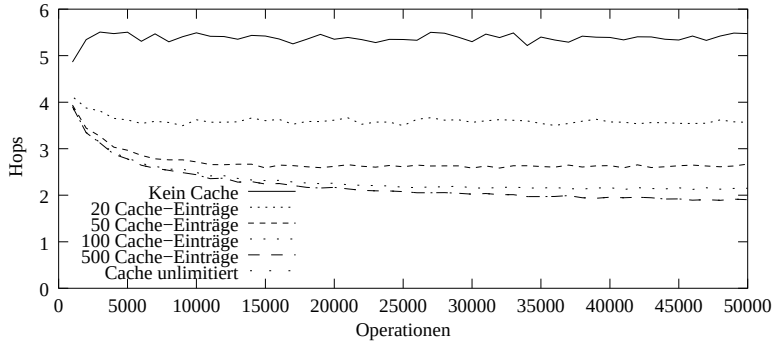


Abbildung 2: Zusammenhang von Cache-Größe zur Anzahl der Message-Hops.

Netz sind also signifikante Verbesserungen zu verzeichnen.

3 Verdrängungsstrategien und lokalitätserhaltende Abbildungen

Nachdem der Nutzen eines Caches für Kontaktinformationen grundsätzlich gezeigt wurde, versuchen wir im Folgenden, dessen Effizienz zu steigern. Naheliegend ist dabei der Einsatz einer besseren Verdrängungsstrategie. Dazu vergleichen wir *Least Frequently Used* (LFU), *Least Recently Used* (LRU), *Least Reference Density* (LRD)[2] mit Random als Referenzstrategie und der speziell auf Routing zugeschnittenen Strategie ISD.

Unsere neue Strategie ISD *(Inverse Square Distribution)* basiert auf einem gleichnamigen Vorschlag zum Aufbau der Weltsicht von Knoten in verteilten Datenstrukturen [Kle00]. Dabei steht p für die Wahrscheinlichkeit zur Aufnahme eines Knotens v in den Cache eines Knotens u. Für die Distanz zwischen zwei Knoten u, v in der Metrik des Schlüsselraums verwenden wir die Notation $d(u, v)$. Der Wert r ist eine Konstante. Es gilt jetzt, dass $p \sim [d(u, v)]^{-r}$. Diese Strategie berücksichtigt das Anfragemuster also nicht. Für $r = d$ mit d als der Dimensionalität des Schlüsselraumes hat Kleinberg unter bestimmten Annahmen gezeigt [Kle00], dass ein dezentraler Algorithmus Pfade von $O(\log n)$ Länge konstruieren kann, wenn die Weltsicht eines Knotens zu o.g. Formel konform ist. Es muss aber noch untersucht werden, ob jene Formel auch zu einer sinnvollen Verdrängungsstrategie führt.

Auswahl der Abbildung auf Schlüsselwerte. Die uns bekannten Schlüsselabbildungen zielen darauf, die zu verwaltenden Daten gleichmäßig über den Schlüsselraum zu verteilen. Ein Ziel dabei ist die gleichmäßige Auslastung der Knoten. In vielen Fällen greifen aufeinanderfolgende Operationen aber auf benachbarte Datenbereiche zu. Die Erhal-

[2]Die *Least Reference Density* (LRD)-Strategie verwendet als Maß für den Nutzen der Cache-Position i die Referenzdichte d_i. Diese wird nach der Gleichung $d_i = \frac{r_i}{g - e_i}$ ermittelt, indem der Referenzzähler r_i durch die Anzahl aller seit dem Einlagerungspunkt e_i des Cache-Elements i registrierten Referenzen g geteilt wird. Damit berücksichtigt diese Strategie zugleich das Alter und die Anzahl der Referenzen eines Eintrags.

tung dieser Lokalität im Schlüsselraum in Verbindung mit Caching von Kontaktdaten führt möglicherweise zu einer deutlichen Verbesserung des Anfrageverhaltens. Wir haben einen Algorithmus erstellt, der Zeichenketten wie URL in einen mehrdimensionalen Schlüsselraum abbildet und diese Lokalität erhalten soll (Abbildung 3). Damit lassen sich Testdaten beispielsweise aus Web-Logs generieren. Der Algorithmus summiert die Zeichencodes eines Strings abwechselnd für jede Dimension auf. Ein mit der Länge der Zeichenkette abnehmender Wichtungsfaktor sorgt dafür, dass Zeichen am Anfang stärker in den resultierenden Schlüsselwert eingehen als die Zeichen am Ende. Der Algorithmus gibt ein Array mit einem n-dimensionalen Punkt – hier im Intervall der Integer-Ganzzahlen – zurück.

```
function {$line} {
  set $faktor = 1000000
  while {[stringlength $line]>0} {
    foreach $i in $dimension {
      set $val($i) = [firstchar $line]*$faktor + $val($i)
      set $line = removefirstchar $line }
    set $faktor = $faktor / 2 }
  return $val() }
```

Abbildung 3: Lokalitätserhaltende Abbildung von Strings in den Schlüsselraum des CAN.

Abbildung 4 stellt exemplarisch den Weg eines Web-Surfers im Schlüsselraum eines CAN dar. Die verwendeten URL sind auf der rechten Seite der Abbildung zu sehen. Das Mapping der URL in den Schlüsselraum erfolgte zum einen gleichverteilt mit einer CRC32-Prüfsumme und zum anderen mit dem Algorithmus gemäß Abbildung 3. Im zweiten Fall werden nur zwei Anfragen langwierig zum Zielknoten geroutet, alle anderen betreffen entweder den letzten Zielknoten oder einen seiner unmittelbaren Nachbarn. Sektion 4 wird quantifizieren, wie sehr die Zahl der Hops durch die Kombination aus lokalitätserhaltender Abbildung und Caching zurückgeht.

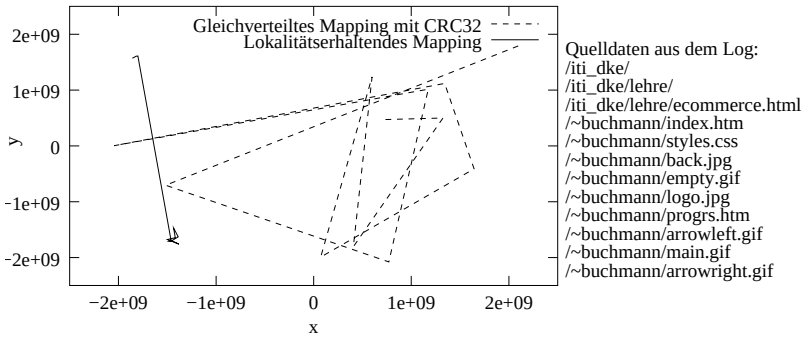


Abbildung 4: Darstellung des Pfades eines Surfers im Schlüsselraum des CAN.

Aktion	Wahrsch.	Abbildung auf das CAN
Link anwählen	52%	Mit 15% Wahrscheinlichkeit wird ein zur letzten Abfrage benachbarter Punkt generiert, mit 85% Wahrscheinlichkeit ein entfernter Punkt.
Back-Button	36%	Es wird der zuvorletzt abgefragte Punkt in der Historie erneut abgefragt.
History oder Bookmark	5%	Ein Punkt aus der Menge der bereits abgefragten wird mit Gleichverteilung zufällig gewählt und erneut abgerufen.
URL eingeben	3%	Ein entfernter Punkt wird zufällig generiert.
Home-Button	2%	Der erste Punkt der Surf-Sitzung wird erneut abgefragt und die Sitzungshistorie gelöscht.
Reload	2%	Es wird der zuletzt abgefragte Punkt in der Historie erneut abgefragt.

Tabelle 1: Verhalten des künstlichen Surfers

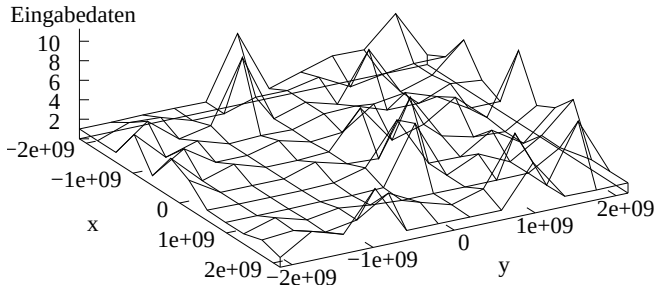


Abbildung 5: Verteilung der künstlich generierten Anfragepunkte im zweidimensionalen Schlüsselraum.

4 Experimente

Experimenteller Aufbau. Es kommen drei Testdatensätze zur Anwendung. Der erste Datensatz enthält gleichverteilte Anfragepunkte und dient als Referenz. Diese Verteilung ist natürlich unrealistisch. Der zweite Datensatz simuliert das Nutzerverhalten von vielen unterschiedlichen Websurfern. Kern der Testdaten-Generierung ist ein künstlicher Surfer zur Erzeugung statistisch relevanter Daten, der das in [TG97] beschriebene Verhalten zeigt. Zunächst wird dabei eine Liste mit über den Schlüsselraum gleichverteilten Startpunkten – entsprechend den Servern im Web – erzeugt. Der nächste Schritt weist jedem Knoten mit Hilfe einer normalverteilten[3] Zufallsfunktion einen dieser Punkte als Startwert zu. Daraufhin werden die Punkte, die jeder einzelne Surfer abfragt, wie in Tabelle 1 dargestellt, sukzessive ermittelt. Abbildung 5 zeigt die Verteilung der Anfragepunkte für ein zweidimensionales CAN. Zu erkennen sind mehrere 'beliebte' Bereiche mit

[3]Da auch im WWW einige Seiten erfolgreicher sind und häufiger abgefragt werden als andere.

hoher Referenzdichte, zahlreiche mäßig und einige schwach nachgefragte Bereiche im Schlüsselraum. Der dritte Datensatz schließlich bildet die Einträge aus dem Access-Log unseres Instituts-Webservers mit einem Algorithmus nach Abbildung 3 auf Testdaten ab. Die Verweildauer und die Zahl der abgesetzten Anfragen der Teilnehmer sind – wahrscheinlich in Folge der unterschiedlichen Interessen von Studenten und Mitarbeitern – verglichen mit dem zweiten Testdatensatz sehr inhomogen: 2.5% aller Teilnehmer generieren ca. 20% aller Anfragen. Als Simulationsumgebung dient wiederum ein CAN mit 1.000 Knoten, in dem die Knoten insgesamt 50.000 Abfragen absetzen.

Auswirkungen der Anfragelokalität. Nachdem Abschnitt 2 die Vorteilhaftigkeit von Caching in unserem Kontext bereits demonstriert hat, soll nun der Nutzen der Lokalitätserhaltung quantifiziert werden. Wir haben Experimente mit unterschiedlichen Cache-Größen durchgeführt, beschränken uns in der Darstellung aber auf die Cache-Größe von 20 Einträgen. Es wird sich herausstellen, dass auch bei sehr kleinen Caches eine lokalitätserhaltende Schlüsselabbildung die Zahl der Hops signifikant verringert. Ohne Caching jedoch würde der Erhalt der Lokalität keinen Nutzen bieten, da auch in diesem Falle stets alle Nachrichten von Nachbar zu Nachbar weitergereicht werden.

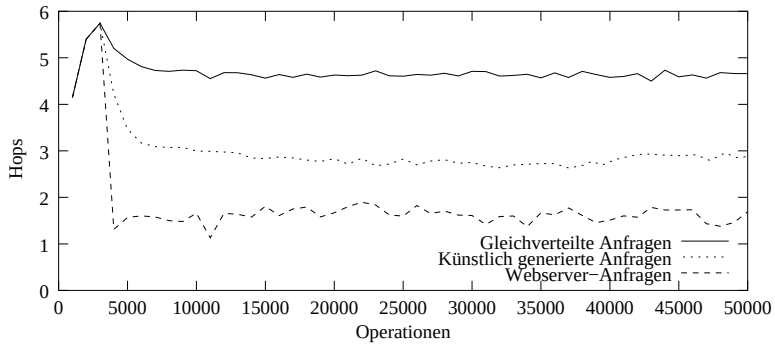


Abbildung 6: Zusammenhang von der Datenverteilung zur Anzahl der Message-Hops.

Abbildung 6 zeigt, dass sich die Effektivität des Caches durch Erhalt der Anfragelokalität erheblich steigern lässt. Diese Steigerung ist jedoch letztendlich trotz unseres recht ausgefeilten experimentellen Aufbaus nur schwer allgemeingültig quantifizierbar, da sie im sehr hohen Maße von den Anfragedaten abhängt. Weil bei den Simulationsdaten, die aus dem Webserver-Log gewonnen wurden, wenige Teilnehmer eine Vielzahl von benachbarten Anfragen stellen, wirkt sich das Caching der Kontaktinformationen erheblich stärker aus als bei einer großen Zahl von aktiven Teilnehmern, wie sie mit dem künstlichen Surfer erzeugt werden.

Einfluß der Verdrängungsstrategie des Cache. Der Cache ist wieder auf 20 Einträge beschränkt. Die Zahl der Knoten im CAN beträgt 3.000. Wir vermuten, dass die Unterschiede zwischen den Verdrängungsstrategien unter beschränkten Verhältnissen besonders

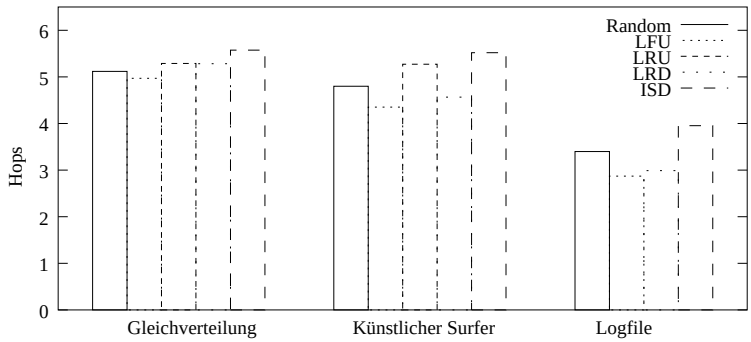


Abbildung 7: Vergleich der Caching-Strategien.

deutlich zu Tage treten. Es werden wieder 50.000 Abfragen abgesetzt, und wir ermitteln den Durchschnitt der Anzahl der Hops aller Nachrichten.

Abbildung 7 zeigt uns, dass die Verdrängungsstrategien nur marginal unterschiedliche Resultate erzielen. Weitere Messungen haben gezeigt, dass bei größeren Caches die Unterschiede weiter verschwinden.

Der Grund für dieses überraschende Resultat ist unseres Erachtens darin zu sehen, wie die Caches mit Daten gefüllt werden. Da die mit einer Weiterleitung überbrückte Distanz mit zunehmender Anzahl der Hops bis zum Erreichen des Ziels asymptotisch gegen Null verläuft, existieren sehr viele Kontakte über kurze Distanz und wenige über lange. Je größer der mit einem Kontakt im Cache überbrückte Schlüsselraum, desto seltener sein Auftreten. Aus diesem Grund erzeugen sämtliche untersuchten Caching-Strategien sehr ähnliche Caching-Inhalte, wenn auch aus völlig unterschiedlichen Positionen. Mit Random kommen häufig vorkommende Kontakte – also diejenigen kurzer Distanz – auch häufiger in den Caches vor. ISD erzielt denselben Effekt durch die Bildung einer expliziten Abhängigkeit von der Distanz der Kontaktinformationen. Da schließlich die häufig vorkommenden Kontaktinformationen kurzer Reichweite auch die von LFU, LRU und LRD geführten Statistiken dominieren, entstehen auch dort ähnliche Verteilungen.

5 Schlußbemerkungen

Verwandte Arbeiten. Dieser Abschnitt geht kurz auf verwandte Arbeiten ein, sofern die Abgrenzung in diesem Artikel noch nicht erfolgt ist.

Vorschläge für verteilte Datenstrukturen existieren in vielen Ausprägungen von verteilten Suchbäumen wie dem Distributed Random Tree [KW94] oder P-Grid [Abe01] bis zu varianten verteilter Hash-Tabellen wie CAN [RFH$^+$01], Chord [SMK$^+$01], LH* [LNS96], Pastry [RD01] oder Tapestry [ZKJ01]. Eine prominente Anwendung dieser verteilten Datenstrukturen sind moderne Peer-to-Peer Netze [MKL$^+$02]. [GHea01] skizziert die Imple-

mentierung von Query-Processing Operatoren in Peer-to-Peer Umgebungen. Auch wenn dieser Artikel im engeren Sinne zu unserem orthogonal ist, zeigt er, dass verteilte, Koordinator-freie Verwaltung großer Datenbestände auch in den Augen anderer wichtig ist.

Das zur Nachrichtenweiterleitung genutzte Verfahren *Greedy Routing* [ADS02] folgt dem *Small-World* Phänomen [Mil67] Milgrams, der durch Experimente zur Weiterleitung von Briefen Eigenschaften sozialer Netze sichtbar machte. Trotz der langen Zeitspanne seit der ersten Publikation Milgrams bestehen jedoch neben den in dieser Arbeit untersuchten Fragen zahlreiche weitere offene Punkte [RSS02] bei der Nachrichtenweiterleitung.

Eine Alternative zur Verbesserung des Routing-Verhaltens besteht darin, explizit leistungsfähige Routen als *Expressways* zu bestimmen. In [XZ02] wird ein Algorithmus vorgestellt, der in einer verteilten Umgebung automatisch Rechner mit hoher freier Kapazität als Endpunkte von Expressrouten bestimmt.

Das Thema von [GBHC00] ist die effiziente Implementierung von SDDS, und die Autoren warten mit eindrucksvollen Performance-Zahlen auf. Allerdings ist die zugrundeliegende Architektur ein zentral administrierter PC-Cluster, also nicht mit der mittelfristigen Ausrichtung unseres Projekts vergleichbar. Außerdem bleibt das Thema Caching bei [GBHC00] explizit außen vor.

Zusammenfassung und Ausblick. Im Rahmen unseres Projekts Fairnet beschäftigen wir uns mit der Weiterentwicklung verteilter skalierbarer Datenstrukturen. Die Voruntersuchungen, die dieser Artikel beschreibt, konzentrieren sich auf zwei Aspekte: Caching der Kontaktinformation und Lokalitätserhaltung der Abbildung der Datenobjekte in den Schlüsselraum der SDDS. Es stellt sich heraus, dass Caching zwar vorteilhaft ist, eine ausgefeilte Caching-Strategie im SDDS-Kontext aber keine Verbesserung mit sich bringt. Die Verwendung einer lokalitätserhaltenden Schlüsselabbildung in Kombination mit Caching hat sich dagegen als sehr vorteilhaft erwiesen. – Zukünftige Untersuchungen werden auch Ausfälle von Knoten betrachten, die die Wirksamkeit des Cachings beeinflussen.

Wir danken Stefan Knöfel, Kathleen Krebs und Mirko Tikalsky für ihre Hilfe bei dieser Studie.

Literaturverzeichnis

[Abe01] Karl Aberer. P-Grid: A Self-Organizing Access Structure for P2P Information Systems. *Lecture Notes in Computer Science*, 2172:179–??, 2001.

[ADS02] James Aspnes, Zoë Diamadi, and Gauri Shah. Fault-tolerant routing in peer-to-peer systems. *21. ACM Symposium on Principles of Distributed Computing*, pages 223–232, July 2002.

[CGM99] Soumen Chakrabarti, David A. Gibson, and Kevin S. McCurley. Surfing the Web backwards. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1679–1693, May 1999.

[GBHC00] Steven D. Gribble, Eric A. Brewer, Joseph M. Hellerstein, and David Culler. Scalable, Distributed Data Structures for Internet Service Construction. In *Proceedings of the*

4th Symposium on Operating Systems Design and Implementation (OSDI-00), pages 319–332, Berkeley, CA, October 23–25 2000. The USENIX Association.

[GHea01] Steven Gribble, Alon Halevy, and Zachary Ives et al. What Can Peer-to-Peer Do for Databases, and Vice Versa? In *Proceedings of the Fourth International Workshop on the Web and Databases*, 2001.

[IRD02] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: A decentralized peer-to-peer web cache. *21th ACM Symposium on Principles of Distributed Computing (PODC 2002)*, 2002.

[Kle00] Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[KW94] Brigitte Kröll and Peter Widmayer. Distributing a Search Tree Among a Growing Number of Processors. In Richard T. Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994*, pages 265–276. ACM Press, 1994.

[LNS96] Witold Litwin, Marie-Anne Neimat, and Donovan A. Schneider. LH* - A Scalable, Distributed Data Structure. *TODS*, 21(4):480–525, 1996.

[Mil67] Stanley Milgram. The small world problem. *Psychology Today*, 1(1):60–67, 1967.

[MKL+02] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Labs, Palo Alto, March 2002.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329 ff., 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In Roch Guerin, editor, *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, volume 31, 4 of *Computer Communication Review*, pages 161–172, New York, August 2001. ACM Press.

[RMW94] M. Roscheisen, C. Mogensen, and T. Winograd. Shared web annotations as a platform for third-party value-added information providers: Architecture, 1994.

[RSS02] Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. Routing Algorithms for DHTs: Some Open Questions. 2002.

[SMK+01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In Roch Guerin, editor, *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, volume 31, 4 of *Computer Communication Review*, pages 149–160, New York, August 27–31 2001. ACM Press.

[TG97] Linda Tauscher and Saul Greenberg. How people revisit web pages: empirical findings and implications for the design of history systems. *International Journal of Human-Computer Studies*, 47(1):97–137, 1997.

[XZ02] Zhichen Xu and Zheng Zhang. Building Low-maintenance Expressways for P2P Systems. Tech Report HPL-2002-41, HP Laboratories, Palo Alto, March 2002.

[ZKJ01] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: an infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley, April 2001.

Kontext-basierte Personalisierung von Web Services

Markus Keidl Stefan Seltzsam Christof König Alfons Kemper

Universität Passau
D-94030 Passau
⟨keidl|seltzsam|koenig|kemper⟩@db.fmi.uni-passau.de

Abstract: Web Services finden zunehmend im Business-To-Consumer-Bereich Verwendung. In diesem Umfeld ist der Kreis der Benutzer heterogen und sehr groß. Jeder hat seine eigene Vorstellung davon, wie der Web Service mit ihm interagieren soll. Web Services müssen deshalb personalisierbar und möglichst flexibel sein, um auf Benutzer in der gewünschten Weise reagieren zu können. In diesem Beitrag präsentieren wir Technologien, die die Entwicklung von derartigen Web Services unterstützen. Mit der dynamischen Dienstauswahl haben Web Services die Möglichkeit, Dienste zur Laufzeit basierend auf einer technischen Spezifikation der gewünschten Dienste auszuwählen und aufzurufen. Mit Vorgaben kann die dynamische Dienstauswahl beeinflusst werden. Vorgaben können direkt beim Aufruf von Diensten angegeben werden oder im Kontext eines Dienstes enthalten sein. Die Verwendung von Kontexten ermöglicht Web Services, Benutzern eine angepasste und personalisierte Version ihrer selbst zur Verfügung zu stellen. Wir präsentieren diese Technologien im Rahmen des ServiceGlobe-Systems, einer Dienstplattform für die flexible und ausfalltolerante Ausführung von Web Services.

1 Einleitung

Web Services haben sich als effizientes Mittel zur Integration von heterogenen Anwendungen im B2B-Bereich (Business-To-Business-Bereich) etabliert. Mit Web Service, im Folgenden auch Dienst genannt, bezeichnen wir eine autonome Softwarekomponente, die durch eine eindeutige URI identifiziert wird und mit den Internet-Standards XML, SOAP oder HTTP angesprochen werden kann. Zunehmend finden Web Services auch im B2C-Bereich (Business-To-Consumer-Bereich) Verwendung. In diesem Umfeld ist der Nutzerkreis heterogener. Er besteht nicht aus einigen Unternehmen, die gegebenenfalls miteinander kooperieren, sondern aus einer großen Menge von Benutzern, jeder mit anderen Vorstellungen davon, wie der Dienst auf ihn reagieren soll. Web Services müssen deshalb personalisierbar und flexibel sein, um mit den Benutzern in der gewünschten Weise interagieren zu können.

In diesem Beitrag präsentieren wir mit der dynamischen Dienstauswahl eine Technologie, die die Entwicklung von personalisierbaren und flexiblen Web Services unterstützt. Sie ermöglicht es Web Services, andere Dienste zur Laufzeit basierend auf einer technischen Spezifikation der gewünschten Dienste auszuwählen und aufzurufen, und stellt damit eine Abstraktion von den tatsächlichen Diensten dar. Mit Vorgaben kann die dynamische

Dienstauswahl beeinflusst werden. Dienste können damit beim Aufruf z. B. anhand ihrer Metadaten ausgewählt werden. Vorgaben können zum einen direkt beim Aufruf von Diensten angegeben werden, sie können aber auch im Kontext eines Dienstes enthalten sein. Unter Kontext verstehen wir dabei Informationen, die ein Web Service benötigt, um Benutzern eine angepasste und personalisierte Version seiner selbst zur Verfügung zu stellen. Die Integration von Vorgaben für die dynamische Dienstauswahl in die Kontexte von Web Services ermöglicht die Personalisierung dieser Web Services, indem die Vorgaben von der Dienstplattform automatisch verwendet werden. Damit kann ein Benutzer einen Web Service z. B. dazu veranlassen, nur kostenlose Dienste aufzurufen.

Wir präsentieren diese Technologien im Rahmen des ServiceGlobe-Systems, einer Dienstplattform für die flexible und ausfalltolerante Ausführung von Web Services. ServiceGlobe unterstützt mobilen Code, d. h. Dienste können zur Laufzeit auf beliebige Internet-Server, die an das ServiceGlobe-System angeschlossen sind, verteilt und dort instanziiert werden. Funktionalität, wie sie für Dienstplattformen Standard ist, z. B. SOAP-basierte Kommunikation, ein Transaktionssystem sowie ein Sicherheitssystem [SBK01], wird ebenfalls unterstützt. Diese Bereiche werden bereits von existierenden Technologien abgedeckt und stellen deshalb nicht den Fokus dieses Beitrags dar. Die Verwendung von ServiceGlobe ermöglicht die Integration der vorgestellten Technologien direkt in die Dienstplattform sowie die Ausnutzung spezifischer ServiceGlobe-Technologien.

Der große Erfolg von Web Services spiegelt sich im kommerziellen Bereich in einer Reihe von Dienstplattformen und Produkten wider, z. B. Sun ONE [Sun], Microsoft .NET [NET] und HP Web Services Platform [WSP]. Diese Produkte und Plattformen basieren auf den bekannten Web-Standards XML [BPSMM00], SOAP [BEK+00], UDDI [UDD00] und WSDL [CCMW01] und bieten Werkzeuge an, um existierende Anwendungen schnell und unkompliziert als Dienste anbieten zu können. Daneben existieren Forschungsplattformen wie ServiceGlobe [KSSK02, KSK02] und SELF-SERV [BDSN02], die sich gezielt auf einzelne Aspekte der Entwicklung und Ausführung von Web Services konzentrieren. Im SELF-SERV-System werden z. B. Dienste mit gleichartigen Schnittstellen zu so genannten Service Communities zusammengefasst. Es wird allerdings nicht näher auf Strategien zur Auswahl von Diensten aus diesen Service Communities eingegangen, wie dies in ServiceGlobe mit der dynamischen Dienstauswahl der Fall ist. Den Schwerpunkt in SELF-SERV bildet vielmehr die Komposition von Web Services mit Hilfe von Statecharts. Weitere Sprachen zur Komposition von Web Services sind IBM's WSFL (Web Services Flow Language) [Ley01], Microsoft's XLang [Tha01] und HP's WSCL (Web Services Conversation Language) [BBB+01]. Compaq's Web Language [Mar99] (ehemals WebL) wurde hauptsächlich zum Laden, Parsen und Generieren von HTML- und XML-Inhalten entwickelt. XL [FK01] dient nicht nur der Komposition von Diensten, sondern unterstützt nach Art einer Programmiersprache auch die Programmierung von Web Services.

Das WWW-Konsortium hat mit [HO02] einen ersten Entwurf vorgelegt, in dem wichtige Anforderungen für Web-Service-Architekturen vorgestellt werden. Hier finden sich z. B. auch Ansätze für eine dynamische Dienstauswahl und die Notwendigkeit von Dienst-Metadaten. Microsoft's Passport [Pas] oder das Liberty Alliance Project [Pro] stellen erste Ansätze für die Nutzung von Kontextinformationen für Web Services dar, wobei zur Zeit nur die Identität eines Benutzers berücksichtigt wird.

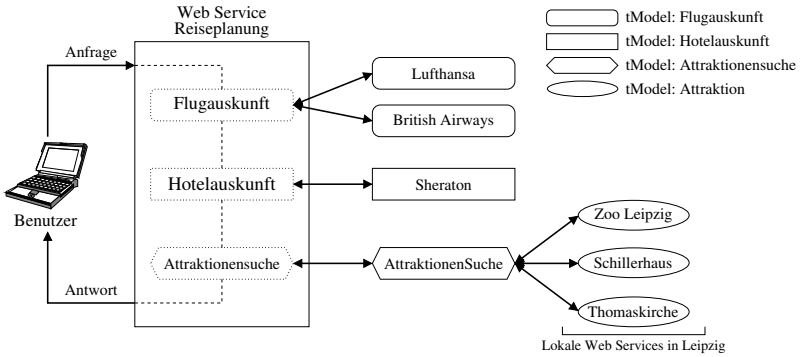


Abbildung 1: Anwendungsszenario: Marktplatz für Reiseagenturen

Der Rest dieses Beitrags ist wie folgt gegliedert: Abschnitt 2 beschreibt ein Anwendungsszenario, das als durchgängiges Beispiel verwendet wird. Im Anschluss präsentiert Abschnitt 3 einen Überblick über die Dienstplattform ServiceGlobe. Abschnitt 4 erläutert das Konzept der dynamischen Dienstauswahl und wie Vorgaben eingesetzt werden können, um die Dienstauswahl zu beeinflussen. Abschnitt 5 beschreibt den Einsatz von Kontext für Web Services im ServiceGlobe-System. Abschnitt 6 beschließt diesen Beitrag mit einer Zusammenfassung.

2 Anwendungsszenario

Im Rahmen dieses Beitrags verwenden wir einen Marktplatz für Reiseagenturen als durchgängiges Anwendungsszenario. Der Marktplatz offeriert Reiseagenturen, aber auch privaten Kunden, die Möglichkeit, komplette Reisen zu planen und zu buchen, inklusive Flug, Hotel, Sehenswürdigkeiten usw. Die Teilnehmer des Marktplatzes (Fluglinien, Hotelketten, etc.) bieten ihre Dienstleistungen als Web Services an. Die Integration der Teilnehmer erfolgt mit Hilfe eines lokalen UDDI-Verzeichnisses, in das die Web Services aller Teilnehmer eingetragen werden. Darin sind Dienste mit der gleichen Semantik einer gemeinsamen technischen Spezifikation zugeordnet, einem sogenannten *tModel* (Abkürzung für „technical model“). Das tModel als technische Spezifikation enthält eine Klassifikation der Funktionalität des Dienstes und eine formale Beschreibung seiner Schnittstelle. Weitere Informationen zu UDDI und tModels findet man z. B. in [RV02].

Der Marktplatz stellt eine Reihe von eigenen Diensten zur Verfügung, die Dienste unterschiedlicher Teilnehmer kombinieren, um damit eine erweiterte Funktionalität und weitergehende Informationen anbieten zu können. Ein Beispiel ist der Dienst *Reiseplanung*, der die automatische Planung einer Urlaubs- oder Dienstreise ermöglicht. Abbildung 1 zeigt eine exemplarische Ausführung des Dienstes für die Planung einer Reise zur BTW-Konferenz 2003 nach Leipzig. Die Ausführung des Dienstes gliedert sich in drei Phasen:[1] Nach dem Erhalt einer Anfrage werden zuerst mehrere Flugauskunftsdienste parallel nach

[1] Das Beispiel betrachtet nur die Suche nach den gewünschten Informationen zu Flügen, Hotels usw. Details zu Buchungen sowie weitere Schritte, die ein vollständiger Dienst zur Reiseplanung durchführen müsste, wurden aus Gründen der Übersichtlichkeit weggelassen.

geeigneten Flügen befragt. Dabei wird ausgenutzt, dass in UDDI Dienste mit der gleichen Semantik einem gemeinsamen tModel zugeordnet sind. Im Beispiel sind alle Flugauskunftsdienste (*Lufthansa* und *British-Airways*) dem tModel `Flugauskunft` zugeordnet. Der Dienst *Reiseplanung* gibt nur an, dass Dienste des tModels `Flugauskunft` aufgerufen werden sollen; die Dienstplattform setzt dies mit Hilfe von UDDI in den Aufruf der tatsächlichen Dienste um. Dieser Vorgang wird im Folgenden als dynamische Dienstauswahl bezeichnet. In Abbildung 1 wird die dynamische Dienstauswahl dadurch verdeutlicht, dass das aufgerufene tModel und die tatsächlich aufgerufenen Dienste durch dieselben Symbole dargestellt werden. Beim Aufruf des tModels wird das Symbol mit einer gepunkteten Linie gezeichnet, bei den aufgerufenen Diensten mit einer durchgehenden Linie. In der zweiten Phase sucht der Dienst *Reiseplanung* nach passenden Übernachtungsmöglichkeiten. Dazu wird ein Aufruf des tModels `Hotelauskunft` ausgeführt, woraufhin die Dienstplattform passende Dienste auswählt und anspricht. In der letzten Phase sucht der Dienst nach Sehenswürdigkeiten vor Ort. Beim Aufruf des tModels `Attraktionensuche` wählt die Dienstplattform den Dienst *AttraktionenSuche* aus, der wie *Reiseplanung* vom Marktplatz zur Verfügung gestellt wird, und spricht ihn an, um von ihm die gewünschte Liste an Sehenswürdigkeiten zu erhalten. Dieser Dienst wiederum führt selbst einen tModel-Aufruf aus (tModel `Attraktion`, aus Gründen der Übersichtlichkeit nicht gezeigt) und spricht lokale Dienste für Sehenswürdigkeiten in und um Leipzig an, um nähere Informationen, z. B. aktuelle Programme oder Öffnungszeiten, einzuholen. In den folgenden Abschnitten werden gezielt verschiedene Punkte dieses Anwendungsszenarios angesprochen und konkretisiert.

3 Die Dienstplattform ServiceGlobe

ServiceGlobe ist eine verteilte, erweiterbare Dienstplattform für die flexible und robuste Ausführung von Web Services. Es ist vollständig in Java 2 implementiert und basiert auf den Standards XML, SOAP, UDDI und WSDL. In diesem Abschnitt werden die grundlegenden Komponenten dieser Plattform präsentiert. In ServiceGlobe werden zwei Arten von Diensten unterschieden: externe und interne Dienste (siehe Abbildung 2).

Externe Dienste sind Dienste wie sie zur Zeit im Internet eingesetzt werden. Sie werden nicht von ServiceGlobe selbst bereitgestellt. Diese Dienste sind normalerweise stationär, d. h. sie werden auf einem bestimmten, festen Rechner ausgeführt. Sie sind auf Basis unterschiedlicher Systeme entwickelt worden und haben verschiedene Aufrufschnittstellen. Um diese Dienste unabhängig von ihrer tatsächlichen Aufrufschnittstelle, z. B. HTML-Formularen oder RPC (Remote Procedure Call), integrieren zu können, verwendet ServiceGlobe *Adaptoren*. Diese setzen interne Anfragen auf die externe Schnittstelle um und umgekehrt. Damit ist auch der Zugriff auf beliebige Anwendungen, z. B. ERP-Systeme (Enterprise Resource Planning), möglich. Externe Dienste können somit genau wie interne Dienste benutzt werden.

Interne Dienste sind Dienste, die auf der Basis der ServiceGlobe-API implementiert wurden. Zu ihrer Entwicklung können auch spezialisierte Programmiersprachen verwendet werden, die dann in native ServiceGlobe-Dienste übersetzt werden müssen. Die Kommunikation interner ServiceGlobe-Dienste wird mit Hilfe von SOAP abgewickelt.

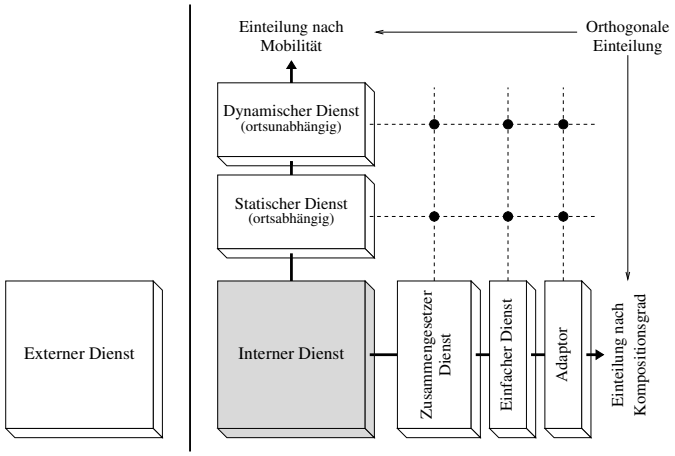


Abbildung 2: Einteilung von Diensten in ServiceGlobe

Dienste erhalten ein einzelnes XML-Dokument als Eingabeparameter und sie generieren ein einzelnes XML-Dokument als Ergebnis. Es werden zwei Arten von internen Diensten unterschieden: *dynamische* und *statische* Dienste. Statische Dienste sind *ortsabhängig*, d. h. sie können nicht dynamisch auf beliebigen ServiceGlobe-Rechnern ausgeführt werden. Gründe dafür können sein, dass sie Zugriff auf gewisse lokale Ressourcen, z. B. ein DBMS zum Speichern von Daten, oder bestimmte Rechte, z. B. Zugriff auf das Dateisystem, benötigten, die nur auf bestimmten, festen Rechnern verfügbar sind. Diese Einschränkungen verhindern die Ausführung von statischen Diensten auf beliebigen ServiceGlobe-Rechnern. Im Gegensatz dazu sind dynamische Dienste *ortsunabhängig*. Sie sind zustandslos, d. h. der interne Zustand eines solchen Dienstes wird nach der Abarbeitung einer Anfrage gelöscht. Sie benötigen keine speziellen Ressourcen oder Rechte und können deshalb auf jedem beliebigen ServiceGlobe-Rechner ausgeführt werden.

Eine orthogonale Einteilung von internen Diensten unterscheidet *Adaptoren*, *einfache Dienste* und *zusammengesetzte Dienste*. Adaptoren wurden bereits definiert. Einfache Dienste sind interne Dienste, die keinen anderen Dienst benutzen. Zusammengesetzte Dienste sind höherwertige Dienste, die aus anderen Diensten zusammengebaut werden. Sie basieren sozusagen auf diesen anderen Diensten, weswegen diese auch als *Basisdienste* bezeichnet werden. Natürlich kann auch ein zusammengesetzter Dienst wiederum als Basisdienst eines anderen zusammengesetzten Dienstes verwendet werden.

Interne Dienste werden auf *Service-Hosts* ausgeführt. Dabei handelt es sich um Rechner, auf denen die ServiceGlobe-Laufzeitumgebung läuft. Die internen Dienste von ServiceGlobe sind mobiler Code. Ihr ausführbarer Code wird, wenn notwendig, von sogenannten *Code-Repositories* auf einen Service-Host geladen. Ein UDDI-Verzeichnis wird verwendet, um Code-Repositories zu finden, von denen ein bestimmter Dienst geladen werden kann. Service-Hosts bieten einen SOAP-Dienst namens *Dynamic Runtime Loading* an, um beliebige dynamische Dienste auszuführen. Dadurch ist die Menge an verfügbaren Diensten nicht fest, sondern sie kann zur Laufzeit von jedem Teilnehmer des

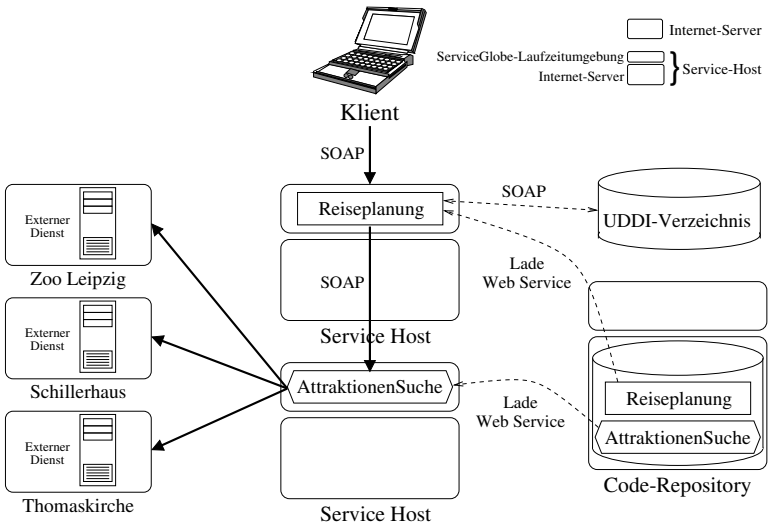


Abbildung 3: Architektur des ServiceGlobe-Systems

ServiceGlobe-Systems erweitert werden. Wenn interne Dienste die notwendigen Rechte besitzen, können sie die Ressourcen eines Service-Hosts benutzen, z. B. lokale Datenbanken. Diese Rechte sind Teil des Sicherheitssystems von ServiceGlobe, das auf [SBK01] basiert, und sie werden autonom von den Administratoren der Service-Hosts verwaltet. Das Sicherheitssystem schützt Service-Hosts auch vor Diensten, die als mobiler Code zur Laufzeit geladen werden und bei denen es sich potenziell um gefährliche Angreifer handeln kann.

Das Laden von Diensten zur Laufzeit ermöglicht die Verteilung von dynamischen Diensten zu beliebigen Service-Hosts. Dadurch ergeben sich für ServiceGlobe eine Reihe von Optimierungsmöglichkeiten: Mehrere Instanzen eines dynamischen Dienstes können aus Gründen der Lastbalancierung und Parallelisierung auf verschiedenen Rechnern ausgeführt werden. Dynamische Dienste können auf Service-Hosts instanziiert werden, die eine möglichst optimale Umgebung für die Ausführung besitzen, z. B. einen schnellen Prozessor, ausreichend Hauptspeicher oder eine schnelle Netzwerkanbindung. Außerdem trägt die Verteilung von dynamischen Diensten zur einer robusten Ausführung bei, da nicht verfügbare Service-Hosts dynamisch durch verfügbare ersetzt werden können.

Abbildung 3 zeigt die wesentlichen Komponenten des ServiceGlobe-Systems und wie sie miteinander interagieren (basierend auf dem Anwendungsszenario aus Abschnitt 2): *Reiseplanung* und *AttraktionenSuche* sind beides dynamische Dienste des Marktplatzes. Die Dienste *Zoo Leipzig*, *Schillerhaus* und *Thomaskirche* sind externe Dienste (Adaptoren wurden in der Abbildung weggelassen). Zuerst sendet der Klient eine SOAP-Anfrage an einen Service-Host des Marktplatzes, den Dienst *Reiseplanung* auszuführen. Der Dienst wird vom Code-Repository geladen (falls er nicht bereits gepuffert wird) und auf dem Service-Host instanziiert. *Reiseplanung* greift während seiner Ausführung auf den dynamischen Dienst *AttraktionenSuche* zu. Mit Hilfe von UDDI wird zuerst ein passender

Service-Host gesucht – im Beispiel ein Service-Host in oder um Leipzig – und der Dienst anschließend im Auftrag des Dienstes *Reiseplanung* auf diesem instanziiert. Der Dienst *AttraktionenSuche* benutzt die drei externen Dienste, um die gewünschten Informationen über Sehenswürdigkeiten zu erhalten.

4 Dynamische Dienstauswahl

Zusammengesetzte Dienste rufen andere Dienste normalerweise auf, indem sie die URL oder den Zugriffspunkt des aufzurufenden Dienstes angeben. Im Gegensatz dazu wird bei der dynamischen Dienstauswahl nur eine technische Spezifikation des Dienstes angegeben, der aufgerufen werden soll. Die Dienstplattform wählt daraufhin geeignete Dienste aus, gegebenenfalls unter Einsatz eines Dienstverzeichnisses wie UDDI, und ruft sie auf. Zusätzlich zur technischen Spezifikation können verschiedene Arten von Vorgaben übergeben werden, um die dynamische Dienstauswahl zu beeinflussen. Das Konzept der dynamischen Dienstauswahl hat drei wichtige Vorteile:

- In verteilten Systemen sind hohe Verfügbarkeit und Fehlertoleranz wichtige Ziele. Bei der Verwendung von zusammengesetzten Diensten kann es passieren, dass einige Basisdienste nicht erreichbar sind, z. B. aufgrund einer Netzwerkpartitionierung, nicht verfügbarer Rechner oder dem Absturz eines Basisdienstes.[2] Aus diesem Grund ist die Verwendung von festen Zugriffspunkten in zusammengesetzten Diensten nicht wünschenswert.
- Die Verwendung von Vorgaben zur Beeinflussung der dynamischen Dienstauswahl hilft bei der Entwicklung von flexiblen Web Services. Wie weiter unten gezeigt wird, ist es nicht notwendig bestimmte Eigenschaften der aufzurufenden Dienste in den zusammengesetzten Dienst selbst zu codieren. Die Eigenschaften werden stattdessen als (deklarative) Vorgaben spezifiziert und dem Dienst zur Laufzeit übergeben. Neu entwickelte Vorgaben können verwendet werden, ohne den Dienst ändern oder neu compilieren zu müssen.
- Dynamische Dienste werden zur Laufzeit instanziiert. Zusammen mit der dynamischen Dienstauswahl ergeben sich eine Vielzahl von Optimierungsmöglichkeiten. Zum Beispiel kann mit Vorgaben festgelegt werden, dass dynamische Dienste im lokalen LAN instanziiert werden müssen oder innerhalb einer festgelegten Menge von Service-Hosts.

Im Folgenden wird die dynamische Dienstauswahl genauer erklärt. Dabei wird UDDI als Dienstverzeichnis verwendet, vor allem weil es sich dabei um den De-Facto-Standard für diese Art von Verzeichnis handelt und weil es die benötigte Funktionalität besitzt.

In UDDI wird jeder Dienst einem tModel zugeordnet.[3] Das tModel als technische Spezifikation enthält eine Klassifikation der Funktionalität des Dienstes und eine formale Be-

[2] Ähnliche Probleme, wenn auch im Zusammenhang mit Web-Scripting-Sprachen, wurden in [CD99a] untersucht.

[3] Tatsächlich kann ein Dienst sogar mehreren tModels zugeordnet werden. Der Aufruf mehrerer tModels unterscheidet sich aber konzeptuell nicht wesentlich vom Aufruf eines tModels und wird deshalb im Folgenden nicht weiter betrachtet.

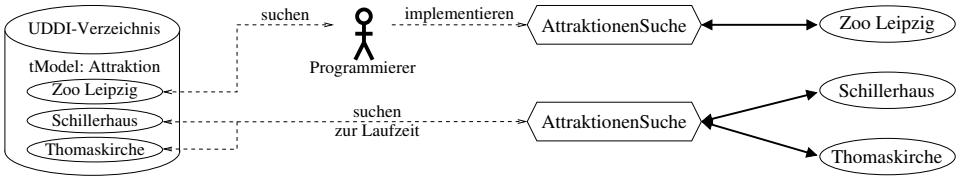


Abbildung 4: Ein Beispiel für dynamische Dienstauswahl

schreibung seiner Schnittstelle. Ein Dienst ist somit eine *Implementierung* oder *Instanz* seines tModels. Statt also in einem Web Service explizit einen Zugriffspunkt anzugeben, wird das tModel angegeben oder „aufgerufen". Dadurch legt man die Funktionalität des Dienstes fest, der aufgerufen werden soll, anstatt eines tatsächlichen Dienstes. Abbildung 4 zeigt ein Beispiel: Dem tModel `Attraktion` sind drei Dienste zugeordnet: *Zoo Leipzig*, *Schillerhaus* und *Thomaskirche*. Angenommen, ein Programmierer will einen neuen Dienst namens *AttraktionenSuche* implementieren, der einen dem tModel `Attraktion` zugeordneten Dienst aufrufen soll. Normalerweise wird der Programmierer im UDDI-Verzeichnis nach einem passenden Dienst suchen, z. B. *Zoo Leipzig*, und dessen Zugriffspunkt im neuen Dienst verwenden. Mit dynamischer Dienstauswahl wird der Programmierer hingegen den unteren der beiden *AttraktionenSuche*-Dienste implementieren. Dieser wird keinen fest codierten Zugriffspunkt enthalten, sondern einen Aufruf des tModels `Attraktion`. Zur Laufzeit wird die Dienstplattform im Auftrag des Dienstes das UDDI-Verzeichnis nach einem geeigneten Dienst durchsuchen und ihn aufrufen (oder sogar mehrere geeignete Dienste, wie in Abbildung 4 gezeigt).

4.1 Vorgaben

Die Vorgaben zur Beeinflussung der dynamischen Dienstauswahl für den Aufruf eines tModels können auf zwei Arten übergeben werden: innerhalb des Kontextes des Dienstes oder direkt beim Aufruf. Auf die erste Art wird in Abschnitt 5 genauer eingegangen. Werden Vorgaben auf die zweite Art übergeben bedeutet dies, dass der Programmierer diese Vorgaben bei der Entwicklung des Dienstes festgelegt hat und dass sie im Code des Dienstes gespeichert sind. Sie können nachträglich nicht mehr geändert werden und sind für jede Ausführung des Dienstes gleich. Eine Änderung ist nur durch eine erneute Compilierung des Dienstes möglich.

4.1.1 Präferenzen und Einschränkungen

Es gibt zwei unterschiedliche Arten von Vorgaben: *Präferenzen* und *Einschränkungen*.[4] Einschränkungen müssen erfüllt werden, wohingegen Präferenzen erfüllt werden sollten. Präferenzen sind also Soll-Vorgaben, d. h. die Dienstplattform wird bei der dynamischen Dienstauswahl als erstes solche Dienste aufrufen, die die Präferenzen erfüllen. Sollte es zu wenig passende Dienste geben, dürfen auch alternative Dienste aufgerufen werden, die die Präferenzen nicht erfüllen. Einschränkungen hingegen sind Muss-Vorgaben, d. h. ein

[4] Eine ähnliche Unterscheidung bei Bedingungen in SQL-Anfragen in hard constraints und soft constraints findet sich in [Kie02].

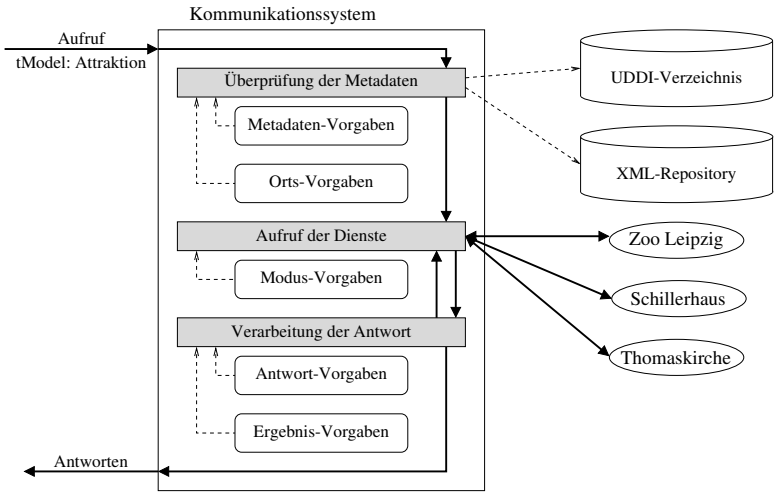


Abbildung 5: Ablauf der dynamischen Dienstauswahl

aufgerufener Dienst muss diese Vorgaben erfüllen. Sind zu wenig passende Dienste vorhanden, können keine alternativen Dienste aufgerufen werden.

4.1.2 Vorgabetypen

Neben der Einteilung in Präferenzen und Einschränkung werden orthogonal dazu fünf verschiedene Typen von Vorgaben unterschieden: Metadaten-Vorgaben, Orts-Vorgaben, Modus-Vorgaben, Antwort-Vorgaben und Ergebnis-Vorgaben. Jeder Vorgabentypen beeinflusst eine bestimmte Phase der dynamischen Dienstauswahl (siehe Abbildung 5). Die Vorgaben eines Typs können wiederum in Präferenzen und Einschränkungen unterschieden werden; bei Modus- und Ergebnis-Vorgaben sind Präferenzen allerdings nicht sinnvoll. Eine Ausnahme bei den Vorgabetypen nehmen die Orts-Vorgaben ein. Sie beeinflussen nicht nur die Auswahl und den Aufruf von Diensten, sondern sie können auch die Auswahl der Service-Hosts beeinflussen, auf denen dynamische Dienste gestartet werden.

Metadaten-Vorgaben Metadaten-Vorgaben (Metadata Constraints) werden vor dem Aufruf von Diensten als Filter auf alle Dienste angewendet, die von UDDI zurückgegeben werden, wenn eine Dienstplattform nach allen zu einem tModel passenden Diensten fragt (wie in Abbildung 5 dargestellt). Metadaten-Vorgaben sind im Grunde XPath-Anfragen [CD99b], die auf die Metadaten eines Dienstes angewendet werden. Ist die Anfrage erfolgreich, kann der Dienst weiterverwendet werden, ansonsten wird er verworfen. Die Metadaten eines einzelnen Dienstes bestehen zum einen aus den Informationen, die in UDDI über ihn zur Verfügung stehen. Das sind neben den bindingTemplate-, businessService- und businessEntity-Einträgen, die auf den Dienst verweisen, auch die tModel-Einträge, die der Dienst referenziert. Dazu kommen zusätzliche Metadaten, die in anderen Metadaten-Repositories gespeichert sind, z. B. [KKKK01, KKKK02b, KKKK02a]. Diese zusätzlichen Metadaten enthalten Informationen, die nicht in UDDI

zu finden sind. Beispiele sind die Kosten für den Aufruf des Dienstes oder das vom Dienst unterstützte Transaktionsprotokoll. Die Metadaten über einen Dienst werden in einem virtuellen XML-Dokument zusammengefasst, auf das die XPath-Anfragen der Metadaten-Vorgaben angewendet werden. Die folgende Metadaten-Präferenz kann beispielsweise verwendet werden, um bevorzugt Dienste der Hotelkette Sheraton auszuwählen:

```
<metadataPreference>
  /businessEntity/name="Sheraton"
</metadataPreference>
```

Genaugenommen werden damit Dienste mit einem businessEntity-Eintrag bevorzugt, dessen Unterelement `name` den Wert `Sheraton` besitzt. Andere Dienste werden nur im Notfall ausgewählt. Die folgende Einschränkung legt fest, dass die Dienstplattform nur kostenlose Dienste auswählen darf (und keinen einzigen kostenpflichtigen Dienst):

```
<metadataCondition>
  /ServiceMetadata/CostsPerCall="0"
</metadataCondition>
```

Orts-Vorgaben Orts-Vorgaben (Location Constraints) werden dazu verwendet, Vorgaben bezüglich des Ausführungsorts eines Dienstes, d. h. des Service-Hosts, zu machen, sowohl für statische als auch für dynamische Dienste. Bei statischen Diensten ermöglicht eine Orts-Vorgabe die Auswahl anhand des Dienststandorts. Bei dynamischen Diensten bewirkt sie, dass diese (im Falle einer Präferenz) bevorzugt oder (im Falle einer Einschränkung) strikt an dem angegebenen Standort instanziiert und ausgeführt werden. Die Informationen über den Standort von Diensten und Service-Hosts erhält die Dienstplattform aus den entsprechenden Metadaten aus dem UDDI-Verzeichnis (siehe Abbildung 5). Der gewünschte Standort kann auf zwei Arten festgelegt werden: basierend auf der Netzwerkadresse des Rechners, auf dem der Dienst läuft oder laufen soll, oder geographisch. Die Unterscheidung in Orts-Vorgaben für statische Dienste und Orts-Vorgaben für dynamische Dienste hat den Vorteil, dass beim Aufruf eines tModels statische und dynamische Dienste unterschiedlich behandelt werden können. Ansonsten wäre es nicht möglich den Standort von statischen Diensten unabhängig vom Standort der Service-Hosts zu wählen, auf denen die dynamischen Dienste ausgeführt werden sollen.

Die folgende Orts-Vorgabe legt fest, dass ausgewählte Dienste in einem Umkreis von 50 Kilometern um Leipzig (ISO-3166-Kürzel DE-SN-LEJ) liegen müssen bzw. dort instanziiert werden sollen. Das Attribut `serviceType` legt fest, dass es sich um eine Orts-Vorgabe sowohl für statische als auch dynamische Dienste handelt. Alternative Werte wären `Static` für statische Dienste und `Dynamic` für dynamische Dienste.

```
<locationCondition addressType="Geographical" serviceType="All">
  <center>DE-SN-LEJ</center>
  <maxDistance>50km</maxDistance>
</locationCondition>
```

Modus-Vorgaben Eine Dienstplattform ist nicht darauf beschränkt beim Aufruf eines tModels nur eine Instanz des tModels auszuwählen; es können auch mehrere Instanzen ausgewählt werden. Somit wird der Aufruf eines tModels ersetzt durch den Aufruf eines oder mehrerer Dienste. Modus-Vorgaben (Mode Constraints) ermöglichen es, die Anzahl

der Dienste festzulegen, die bei einem tModel-Aufruf angesprochen werden sollen. Wie Abbildung 5 zeigt, sind Modus-Vorgaben entscheidend für den Aufruf der Dienste und die Verarbeitung ihrer Antworten. Aufgrund der Modus-Vorgaben wird entschieden, ob nach der Verarbeitung einer Antwort ein alternativer Dienst aufgerufen werden soll, ob alle Antworten als Ergebnis zurückgegeben werden sollen oder ob auf weitere Antworten gewartet werden muss. Die folgenden Modi werden beim Aufruf eines tModels unterschieden: One-Modus, Some-Modus und All-Modus.[5]

Im *One-Modus* wird nur eine Instanz des tModels aufgerufen. Im Falle eines Fehlers, z. B. wenn der Dienst zeitweilig nicht verfügbar ist, wird ein alternativer Dienst aufgerufen. Im *Some-Modus* wird eine Teilmenge der von UDDI zurückgelieferten Dienste aufgerufen. Die Größe der Menge, d. h. die Anzahl der aufzurufenden Dienste, wird als Parameter angegeben (als Prozentangabe oder absolut). Dienste, die nicht antworten, werden durch alternative Dienste ersetzt, solange bis die geforderte Anzahl von Diensten erfolgreich geantwortet hat oder keine alternativen Dienste mehr verfügbar sind. Im *All-Modus* werden alle zurückgelieferten Dienste aufgerufen. Bei Fehlern können natürlich keine alternativen Dienste aufgerufen werden.

Das folgende Beispiel zeigt eine Some-Modus-Vorgabe, die festlegt, dass fünf Prozent der verfügbaren Dienste aufgerufen werden sollen:

```
<modeCondition modeType="Some" number="5" numberType="Percentage" />
```

Antwort-Vorgaben Antwort-Vorgaben (Reply Constraints) werden ausgewertet, wenn eine Antwort von einem aufgerufenen Dienst eintrifft (siehe Abbildung 5). Eine Antwort wird nur an den aufrufenden Dienst zurückgegeben, wenn sie alle für sie relevanten Antwort-Vorgaben erfüllt, ansonsten wird sie verworfen. Es gibt zwei Arten von Antwort-Vorgaben: Eigenschafts-Vorgaben und Selektions-Vorgaben.

Eine *Eigenschafts-Vorgabe* (Property Constraint) erlaubt die Auswahl von Antworten basierend auf einer oder mehrerer, fest definierter Eigenschaften der Antwort. Die Eigenschaften müssen entweder von der Dienstplattform oder aber vom aufgerufenen Dienst zur Verfügung gestellt werden. Ein Dienst erreicht dies dadurch, dass er entsprechende XML-Elemente in seine Antwort einfügt. ServiceGlobe unterstützt folgende Eigenschaften: Verschlüsselung, Signatur und Alter der Daten. Mit den ersten beiden kann überprüft werden, ob eine Antwort verschlüsselt bzw. signiert ist und von wem sie signiert ist. Die letzte Eigenschaft kann verwendet werden, um das Alter der gelieferten Daten zu überprüfen. Wichtig ist dies zum Beispiel, wenn die Antwort aus einem Cache stammt, man aber ausreichend aktuelle Daten haben möchte. Die folgende Vorgabe würde eine Dienstplattform dazu veranlassen, nur Antworten zurückzugeben, die vom angegebenen Subjekt signiert sind:

```
<propertyCondition>
  <signature>
    <signatureDN>CN=Customer, O=Universität Passau, C=DE</signatureDN>
  </signature>
</propertyCondition>
```

[5] Die Modi entsprechen den Kommunikationsformen Unicast, Multicast und Broadcast in Netzwerken

Selektions-Vorgaben (Selection Constraints) erlauben die Anwendung eines beliebigen XPath-Ausdrucks auf die Antwort eines Dienstes, inklusive ihrer SOAP-spezifischen Teile, wie z. B. den SOAP-Header. Verschlüsselte Teile der Nachricht können natürlich nicht abgefragt werden. Selektions-Vorgaben, die auf verschlüsselte Teile Bezug nehmen, gelten als nicht erfüllt, was bei Einschränkungen dazu führt, dass die Antwort nicht an den aufrufenden Dienst zurückgegeben wird. Die folgende Vorgabe stellt beispielsweise sicher, dass alle Antworten im SOAP-Body ein Unterelement `Hotel` mit einem Attribut `class` und dem Wert `FirstClass` enthalten:

```
<selectionCondition>
  /Envelope/Body//Hotel/@class="FirstClass"
</selectionCondition>
```

Ergebnis-Vorgaben Der fünfte Vorgabentyp sind Ergebnis-Vorgaben (Result Constraints). Anders als Antwort-Vorgaben betreffen sie keine einzelne Antwort, sondern alle bisher eingetroffenen Antworten. Es gibt zwei Arten von Ergebnis-Vorgaben: Timeout-Vorgaben und FirstN-Vorgaben.

Mit einer *Timeout-Vorgabe* (Timeout Constraint) kann eine maximale Wartezeit für Antworten von aufgerufenen Diensten festgelegt werden. Nach Ablauf dieser Zeitspanne werden alle noch ausstehenden Dienste abgebrochen und die bis zu diesem Zeitpunkt erhaltenen Antworten an den aufrufenden Dienst zurückgegeben. Diese müssen natürlich alle anderen relevanten Vorgaben erfüllen, z. B. Antwort-Vorgaben. Folgende Vorgabe ist ein einfaches Beispiel für eine Timeout-Vorgabe:

```
<timeoutCondition value="100" valueUnit="Seconds"/>
```

FirstN-Angaben ermöglichen es, den Aufruf eines tModels nach dem Erhalt einer vorher festgelegten Anzahl von Antworten zu beenden. Der aufrufende Dienst erhält nur diese Antworten zurück. Aufrufe an Dienste, die bis zu diesem Zeitpunkt noch nicht geantwortet haben, werden abgebrochen. Die Anzahl an abzuwartenden Antworten kann entweder absolut (Attribut `numberType="Absolute"`) oder prozentual (`numberType="Percentage"`), abhängig von der Anzahl der initial aufgerufenen Dienste, festgelegt werden. Die folgende Vorgabe würde den Aufruf eines tModels nach dem Eintreffen von zehn Prozent der Antworten beenden:

```
<firstNCondition number="10" numberType="Percentage" />
```

4.2 Kombination von Vorgaben

Um die dynamische Dienstauswahl auf komplexere Weise zu beeinflussen, ist die Kombination mehrerer Vorgaben notwendig. Will man z. B. die Auswahl so beeinflussen, dass nur Dienste einer vorgegebenen Firma aufgerufen werden (Metadaten-Vorgabe) und außerdem maximal 10 Sekunden auf Antworten gewartet wird (Timeout-Vorgabe), benötigt man zwei konjunktiv verknüpfte Vorgaben (AND-Operator).

In ServiceGlobe können Vorgaben mit Hilfe der Operatoren AND und OR kombiniert werden.[6] Für jeden der Operatoren gibt es ein entsprechendes XML-Element: `andGroup`

[6] An der Integration des Operators NOT wird zur Zeit gearbeitet.

```
<orGroup>
  <andGroup>
    <metadataCondition>
      /ServiceMetadata/ServiceType="Dynamic"
    </metadataCondition>
    <locationPreference serviceType="Dynamic"
                        addressType="Geographical">
      <pattern>DE-SN-LEJ</pattern>
    </locationPreference>
  </andGroup>
  <locationPreference serviceType="All" addressType="Geographical">
    <pattern>DE-*-*</pattern>
  </locationPreference>
</orGroup>
```

Abbildung 6: Kombination von Vorgaben

für den AND-Operator und `orGroup` für den OR-Operator. Jeder Operator ermöglicht die Kombination von Vorgaben, sogenannten *atomaren Termen*, und bereits kombinierten Vorgaben, auch *kombinierte Terme* genannt, zu einem neuen (kombinierten) Term. Das bedeutet, dass die Operatoren ineinander geschachtelt werden können. Abbildung 6 zeigt ein Beispiel: die OR-Kombination von zwei Termen. Der erste Term sagt aus, dass nur dynamische Dienste ausgewählt und vorzugsweise auf Service-Hosts in Leipzig instanziiert werden sollen.[7] Der zweite Term legt fest, dass beliebige Dienste in Deutschland bevorzugt ausgewählt werden sollen.

Durch die Kombination von Vorgaben kann es dazu kommen, dass Konflikte entstehen, die die Erfüllung aller Vorgaben verhindern, wie folgendes Beispiel zeigt:

```
<orGroup>
  <metadataCondition>
    /businessEntity/name="Sheraton"
  </metadataCondition>
  <timeoutCondition value="100" valueUnit="Seconds"/>
</orGroup>
```

Bei der Auswertung dieser kombinierten Vorgabe hat die Dienstplattform zwei Möglichkeiten: Sie kann zum einen *nur Dienste des Unternehmens Sheraton* ansprechen und warten bis alle geantwortet haben. In diesem Fall erfüllt sie nur die erste Vorgabe. Oder sie kann zum anderen *alle Dienste* des aufgerufenen tModels ansprechen. Im Falle eines Timeouts muss die Dienstplattform dann entscheiden, ob sie alle bisher erhaltenen Antworten zurückgibt und damit nur die zweite Vorgabe erfüllt oder ob sie den Timeout ignoriert, auf alle Antworten der Sheraton-Dienste wartet und damit nur die erste Vorgabe erfüllt. In diesem Fall hat sie allerdings initial unnötig viele Dienste angesprochen. Im Allgemeinen kann die Dienstplattform also nicht beide Vorgaben gleichzeitig erfüllen.

4.3 Vorgabenkonflikte und Konfliktauflösung

Die Kombination von Vorgaben kann zu verschiedenen Konflikten führen. Ein *Widerspruch* tritt auf, wenn zwei oder mehr Vorgaben angegeben werden, die nicht alle zu-

[7] Mit dem Unterelement `pattern` der Orts-Vorgabe wird ein Muster definiert, zu dem die geographischen Informationen der Service-Hosts passen müssen. Das Zeichen `*` agiert als Wildcard.

```
<dssConstraints>
  <locationPreference srcKey="1" addressType="Geographical">
    <pattern>DE-*-*</pattern>
  </locationPreference>
</dssConstraints>
<dssConstraintsSources>
  <source srcKey="1">
    <URI>http://tempuri.org/sg/constraints</URI>
    <priority>2</priority>
  </source>
</dssConstraintsSources>
```

Abbildung 7: Vorgaben mit Prioritäten und Quellenangaben

gleich erfüllt werden können. Ein Beispiel dafür sind zwei Modus-Einschränkungen, die unterschiedliche Aufrufmodi fordern. Ein weiteres Beispiel sind zwei Metadaten-Einschränkungen, bei denen die eine die Verwendung von dynamischen Diensten fordert, während die andere statische Dienste verlangt. Widersprüche können nur zwischen Einschränkungen auftreten, da Präferenzen im Falle eines Widerspruchs mit einer anderen Vorgabe immer fallen gelassen werden können. Die zweite Art von Konflikt wurde bereits im vorhergehenden Abschnitt erklärt. Bei ihr ist es nicht immer möglich alle Vorgaben zu erfüllen. Die Dienstplattform muss in diesem Fall entscheiden, welche Vorgaben erfüllt werden sollen und welche nicht. Hierbei handelt es sich um eine Frage der Optimierung: Einerseits soll die Dienstplattform möglichst nur so viele Dienste ansprechen wie unbedingt nötig, andererseits sollen so viele Vorgaben erfüllt werden wie möglich.

Die Ursache für derartige Konflikte können Fehler bei der Zusammenstellung der Vorgaben sein. Hauptsächlich werden solche Konflikte aber auftreten, wenn für einen tModel-Aufruf Vorgaben aus unterschiedlichen Quellen verwendet werden, z. B. vom Dienst selbst oder aus seinem Kontext. Eine Dienstplattform muss Konflikte auflösen oder die Abarbeitung der Anfrage abbrechen. Der Rest dieses Abschnitts erläutert, wie eine Dienstplattform Konflikte auflösen und das Abbrechen von Anfragen vermeiden kann.

Die Auflösung von Konflikten basiert auf Prioritäten. Jedem Term wird eine Priorität von 0 (Minimum) bis ∞ (Maximum) zugeordnet. Eine implizite Priorisierung ist durch die Reihenfolge der Terme in ihrer XML-Repräsentation gegeben. Je später ein Term im XML-Dokument definiert wird, desto geringer ist seine Priorität. Eine explizite Priorisierung kann auf zwei Arten durchgeführt werden: Zum einen kann jedem Term mit dem Attribut `priority` eine explizite Priorität zugeordnet werden. Zum anderen kann mit dem Attribut `srcKey` eine Referenz auf die Quelle eines Terms angegeben werden. Den Quellen – identifiziert durch eine URI – werden in einem gesonderten Abschnitt des Kontextes Prioritäten zugeordnet. Diese werden dann auf die jeweiligen Terme angewendet. Die Angabe einer Quelle wirkt sich – bei kombinierten Termen – rekursiv auf alle enthaltenen Teilterme aus, solange bis gegebenenfalls eine weitere Quellenangabe folgt. Abbildung 7 zeigt ein Beispiel. Besitzen Terme die gleiche explizite Priorität, entscheidet ihre implizite Priorität, d. h. die Reihenfolge, in der sie angegeben wurden.

4.4 Auswertung von Vorgaben

Der folgende Abschnitt beschreibt die Vorgehensweise von ServiceGlobe bei der Auswertung von Vorgaben beim Aufruf eines tModels. Sie besteht aus zwei Schritten: Zuerst werden alle für den tModel-Aufruf relevanten Vorgaben zusammengefasst und Konflikte aufgelöst. Anschließend werden den Vorgaben entsprechende Dienste ausgewählt und aufgerufen und ihre Antworten verarbeitet.

Vorverarbeitung der Vorgaben

Zuerst werden alle für einen tModel-Aufruf relevanten Vorgaben aus den verschiedenen Quellen konjunktiv zusammengefasst. Die resultierende, kombinierte Vorgabe wird in ihre disjunktive Normalform umgewandelt. Das Ergebnis ist eine kombinierte Vorgabe der folgenden Form:

$$(c_{11} \wedge \ldots \wedge c_{1i_1}) \vee (c_{21} \wedge \ldots \wedge c_{2i_2}) \vee \ldots \vee (c_{n1} \wedge \ldots \wedge c_{ni_n})$$

mit $\forall k \in \{1, \ldots, n\}, j \in \{1, \ldots, i_k\} : c_{kj} \in$ *Vorgaben*, d. h. die c_{kj} sind atomare Terme. Die Vorgaben in jedem *AND-Term*, d. h. einem Term, der nur $\wedge$-Operatoren enthält, werden anhand ihres Auswertungszeitpunkts sortiert. Die Reihenfolge lautet: Metadaten-Vorgaben, Orts-Vorgaben, Modus-Vorgaben, Antwort-Vorgaben und Ergebnis-Vorgaben. Die AND-Terme der obigen kombinierten Vorgabe seien bereits so geordnet. Damit gilt:

$$\begin{aligned} \forall k \in \{1, \ldots, n\} \quad : \quad & c_{k1}, \ldots, c_{km_k} \in \textit{MetadatenVorgaben} \\ & c_{km_k+1}, \ldots, c_{kl_k} \in \textit{OrtsVorgaben} \\ & c_{kl_k+1}, \ldots, c_{kp_k} \in \textit{ModusVorgaben} \\ & c_{kp_k+1}, \ldots, c_{kr_k} \in \textit{AntwortVorgaben} \\ & c_{kr_k+1}, \ldots, c_{ki_k} \in \textit{ErgebnisVorgaben} \end{aligned}$$

Die Menge der Vorgaben eines Typs für einen einzelnen AND-Term kann natürlich leer sein. Jeder AND-Term darf maximal eine Modus-Vorgabe enthalten, d. h. $\forall k \in \{1, \ldots, n\} : l_k + 1 = p_k$, ansonsten existiert ein Konflikt. Die Auflösung von Konflikten geschieht dadurch, dass aus allen miteinander in Konflikt stehenden Vorgaben diejenige mit der höchsten Priorität ausgewählt wird. Alle anderen Vorgaben werden entfernt. Damit ein Dienst die kombinierte Vorgabe erfüllt, muss er mindestens einen der AND-Terme erfüllen, d. h. die Metadaten des Dienstes, der Modus seines Aufrufs und seine Antwort müssen den Vorgaben des AND-Terms entsprechen.

Aufruf der Web Services

Im nächsten Schritt werden von UDDI Informationen über alle Dienste angefordert, die das aufgerufene tModel implementieren. Diese Dienste werden in einer Liste geordnet. Im Folgenden bewirken Einschränkungen, dass Dienste, die eine Einschränkung nicht erfüllen, verworfen werden. Präferenzen führen dazu, dass die Dienste anhand der erfüllten Präferenzen und deren Prioritäten geordnet werden. Je mehr Präferenzen ein Dienst erfüllt und je höher die Prioritäten der Präferenzen sind, desto weiter vorne steht er in der Liste.

Anschließend werden zuerst Metadaten-Vorgaben angewendet, dann Orts-Vorgaben. Bei Orts-Vorgaben für dynamische Dienste werden zuerst alle Service-Hosts vom UDDI-Verzeichnis angefordert und entsprechend den Orts-Vorgaben gefiltert. Jeder dynamische

Dienst wird dann einem Service-Host aus der Liste zugeteilt. Anschließend werden die Dienste aufgerufen, wobei eine Modus-Vorgabe bestimmt, wieviele Dienste initial aufgerufen werden und ob gegebenenfalls alternative Dienste angesprochen werden müssen. Die Antworten der Dienste werden mit Hilfe der Antwort-Vorgaben gefiltert und wiederum führen Präferenzen zu einer Sortierung der Antworten. Sind alle Ergebnis-Vorgaben erfüllt, werden die bisher gesammelten Antworten unter Berücksichtigung ihrer Sortierung und der Ergebnis-Vorgaben zurückgegeben.

Bei der Auswertung von kombinierten Vorgaben mit mehreren AND-Termen wird momentan der AND-Term mit der höchsten Priorität[8] ausgewählt. In Zukunft werden im Rahmen einer Optimierungskomponente verschiedene Optimierungskriterien berücksichtigt werden und die Auswahl wird darauf basieren.

5 Web-Service-Kontext

Kontext bzw. die in ihm enthaltenen Informationen sind ein effizientes Mittel, um personalisierbare und flexible Web Services zu entwickeln. Gerade die Integration von Vorgaben in den Kontext ist dabei hilfreich. Der folgende Abschnitt beschreibt, wie Kontext in das ServiceGlobe-System integriert ist und welche Informationen er enthalten kann.

5.1 Integration in Dienstplattformen

Kontextinformationen sind nicht Teil der direkten Eingabedaten eines Dienstes; sie stellen zusätzliche Informationen dar, die nicht vorhanden sein müssen. Die Integration des Kontextes erfolgt transparent, d. h. Dienste müssen sich nicht aktiv darum kümmern Kontextinformationen zu verarbeiten, sie in ausgehende Nachrichten einzufügen oder aus eingehenden Nachrichten auszulesen. Dies wird von der Dienstplattform erledigt. Kontextinformationen können je nach Typ von der Dienstplattform oder vom aufgerufenen Dienst ausgewertet werden. Soweit möglich sollten sie von der Dienstplattform automatisch ausgewertet werden, da Dienste nur Kontextinformationen berücksichtigen können, die bei ihrer Implementierung bekannt waren und integriert wurden. Kontextinformationen, die weder von der Dienstplattform noch vom aufgerufenen Dienst unterstützt werden, werden ignoriert. Insbesondere können natürlich existierende Dienste Kontext ohne weiteres ignorieren.

Aus Gründen der Transparenz werden Kontextinformationen beim Aufruf eines Dienstes nicht als Teil des SOAP-Bodies der Nachricht übertragen – dieser enthält die direkten Eingabedaten des Dienstes – sondern als Teil des SOAP-Headers. Dienstplattformen, die nicht mit Kontextinformationen umgehen können, können diesen Teil des Header ignorieren.

[8] Die Priorität eines AND-Terms ist (momentan) das Maximum der Prioritäten seiner Vorgaben.

5.2 Arten von Kontextinformationen

Der folgende Abschnitt beschreibt die verschiedenen Arten von Informationen, die der Kontext eines Web Services in ServiceGlobe enthalten kann.

Kontakt- und Klientinformationen

Kontaktinformationen enthalten Daten über die Identität des Benutzers, z. B. Name, Adresse, Email-Adresse, soweit der Benutzer bereit ist diese anzugeben. Benutzer haben auch die Möglichkeit, Pseudonyme anzugeben oder anonym zu bleiben. Die Daten werden direkt aus dem Klienten entnommen, auf dem der Benutzer die initiale Anfrage gestellt hat. Außerdem können die Kontaktinformationen Daten über Personalisierungsdienste, z. B. das Liberty Alliance Project [Pro] oder Microsoft Passport [Pas], enthalten, die von den Web Services oder der Dienstplattform zur automatischen Identifikation des Benutzers verwendet werden können.

Klientinformationen enthalten Daten zur Hard-/Software des Klienten sowie seinen Standort. Der Standort umfasst nicht nur die geographische Angabe der Position des Benutzers sondern auch semantische Informationen darüber. Der lokalisierte Kontext kann für Optimierungen bei der Dienstausführung und zur Optimierung von UDDI-Anfragen verwendet werden. Informationen über den Klienten werden verwendet, um den Umfang der Antwort des Web Service anzupassen. Je nach Klient können z. B. Bilder entweder direkt oder als Verweis in die Antwort eingebunden werden. Verweise sind sinnvoll, wenn der Benutzer an einem PDA mit einer langsamen oder teuren Netzverbindung arbeitet. Auch die Größe der Bilder kann an den Klienten angepasst werden. Klientinformationen ermöglichen es zudem, in die Antwort die passenden Stylesheets einzubinden, die es dem Klienten ermöglichen, die XML-Antwort entsprechend dem Ausgabemedium zu formatieren, z. B. in HTML-Seiten oder WAP-Seiten.

Vorgaben für die dynamische Dienstauswahl

In Abschnitt 4.1 wurden verschiedene Vorgaben vorgestellt, die bei der dynamischen Dienstauswahl verwendet werden können. Die Angabe von Vorgaben beim Aufruf eines tModels erfordert, dass die Vorgaben im Code des Dienstes enthalten sind und deshalb bereits bei der Compilierung des Dienstes festgelegt wurden. Statt direkt beim Aufruf können Vorgaben auch in den Kontext eines Dienstes eingefügt werden. Beim Aufruf eines tModels bestimmt die Dienstplattform alle relevanten Vorgaben aus dem Kontext und verwendet diese – zusätzlich zu direkt angegebenen Vorgaben. Dadurch ergibt sich eine zusätzliche Flexibilität bei der dynamischen Dienstauswahl, wodurch eine automatische Anpassung der Dienste an die Wünsche der Benutzer, z. B. die Verwendung der aktuell billigsten Dienste, möglich wird. Durch die Einbindung von Vorgaben in den Kontext kann außerdem eine Optimierung der Ausführung der angesprochenen Dienste erreicht werden.

Ein Dienst wird im Allgemeinen mehr als ein tModel aufrufen. Folglich wird der Kontext eines Dienstes Vorgaben für mehr als einen Aufruf enthalten und für die Vorgaben muss deshalb festgelegt werden, für welchen Aufruf sie verwendet werden sollen. Aus diesem Grund kann bei jeder Vorgabe das tModel (mit dem Attribut `tModel`) bzw. der Schlüssel

```
<Context xmlns="http://sg.fmi.uni-passau.de/context">
  <dssConstraints>
    <metadataCondition tModel="Reiseplanung">
      /ServiceMetadata/CostsPerCall="0"
    </metadataCondition>
    <metadataCondition tModel="Attraktionensuche">
      /ServiceMetadata/ServiceType="Dynamic"
    </metadataCondition>
    <locationPreference tModel="Attraktionensuche" serviceType="Dynamic"
                        addressType="Geographical">
      <pattern>DE-SN-LEJ</pattern>
    </locationPreference>
    <locationCondition tModel="Attraktion" addressType="Geographical">
      <center>DE-SN-LEJ</center>
      <maxDistance>50km</maxDistance>
    </locationCondition>
    <propertyCondition tModel="Attraktion">
      <maxAgeOfData>1d</maxAgeOfData>
    </propertyCondition>
  </dssConstraints>
</Context>
```

Abbildung 8: Vorgaben im Kontext eines Dienstes

des tModels (Attribut `tModelKey`) angegeben werden, bei dessen Aufruf die Vorgabe verwendet werden soll. Wird nichts derartiges angegeben, gilt die Vorgabe für alle Aufrufe. Abbildung 8 zeigt ein Beispiel für Vorgaben mit Angabe des gültigen tModels. Auch AND- und OR-Kombinationen können mit den Attributen `tModel` bzw. `tModelKey` einem tModel-Aufruf zugeordnet werden. In einem solchen Fall sind alle in dem Term definierten Vorgaben dem angegebenen tModel-Aufruf zugeordnet. Es ist offensichtlich, dass eine Kombination von Vorgaben, die unterschiedlichen tModel-Aufrufen zugeordnet sind, nicht sinnvoll ist.

Vorgaben im Kontext eines Dienstes können aus mehreren unterschiedlichen Quellen stammen: vom Benutzer, von Dienstplattformen oder von Diensten in der Aufrufkette. Der Klient eines Benutzers hat die Möglichkeit, Vorgaben beim Abschicken einer Anfrage automatisch in den Kontext einzufügen. Diese Vorgaben kann der Benutzer direkt angegeben haben, der Klient kann sie im Laufe der Zeit angesammelt haben – ähnlich wie Bookmarks bei Web-Browsern – oder die Vorgaben werden vom Arbeitgeber des Benutzers vorgegeben. Eine Dienstplattform kann beim Aufruf eines lokalen Dienstes eine Menge von lokalen Standardvorgaben in den Kontext des aufgerufenen Dienstes einfügen. Schließlich kann auch ein Dienst beim Aufruf eines Basisdienstes Vorgaben in die ausgehende Nachricht einfügen. Der aufgerufene Dienst kann beim Aufruf von eigenen Basisdiensten diese Vorgaben weitergeben und gegebenenfalls auch selbst neue einfügen.

6 Zusammenfassung

Wir haben in diesem Beitrag Technologien präsentiert, die in ihrer Kombination die Entwicklung von personalisierbaren und flexiblen Web Services erleichtern. Die dynamische Dienstauswahl ermöglicht die Auswahl und den Aufruf von Web Services zur Laufzeit, ba-

sierend auf einer technischen Spezifikation der gewünschten Web Services. Sie kann durch verschiedene Vorgaben beeinflusst werden, die direkt beim Aufruf der Dienste oder in den Kontexten der Dienste angegeben werden können. Der Einsatz von Kontexten ermöglicht es personalisierbare Web Services zu entwickeln, die mit ihren Benutzern in der von ihnen gewünschten Weise interagieren. Neben Vorgaben kann der Kontext eines Dienstes auch Informationen zum Benutzer oder der Klientumgebung enthalten.

Wir haben diese Technologien im Rahmen des ServiceGlobe-Systems präsentiert, einer Dienstplattform für die flexible und robuste Ausführung von Web Services. Neben den genannten Technologien unterstützt ServiceGlobe mobilen Code, wodurch eine Verteilung der Web Services zur Laufzeit hin zu beliebigen Service-Hosts im Netz möglich wird.

Literaturverzeichnis

[BBB+01] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, and S. Williams. Web Services Conversation Language (WSCL). `http://www.e-speak.hp.com/media/wscl_5_16_01.pdf`, 2001. Hewlett-Packard.

[BDSN02] B. Benatallah, M. Dumas, Q. Z. Sheng, and A. H. H. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In *Proc. of the 18th Intl. Conference on Data Engineering (ICDE)*, pages 297–308, 2002.

[BEK+00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. `http://www.w3.org/TR/SOAP`, 2000. W3C Note.

[BPSMM00] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). `http://www.w3.org/TR/REC-xml`, 2000. W3C Recommendation.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, 2001. W3C Note.

[CD99a] L. Cardelli and R. Davies. Service Combinators for Web Computing. *IEEE Trans. Software Eng.*, 25(3):309–316, 1999.

[CD99b] J. Clark and S. DeRose. XML Path Language (XPath). `http://www.w3.org/TR/xpath`, 1999. W3C Recommendation.

[FK01] D. Florescu and D. Kossmann. An XML Programming Language for Web Service Specification and Composition. *IEEE Data Engineering Bulletin*, 24(2):48–56, 2001.

[HO02] H. Haas and D. Orchard. Web Services Architecture Usage Scenarios. `http://www.w3.org/TR/ws-arch-scenarios`, 2002. W3C Working Draft.

[Kie02] W. Kießling. Foundations of Preferences in Database Systems. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 311–322, 2002.

[KKKK01] M. Keidl, A. Kreutz, A. Kemper, and D. Kossmann. Verteilte Metadatenverwaltung für die Anfragebearbeitung auf Internet-Datenquellen. In *Proc. GI Conf. on Database Systems for Office, Engineering, and Scientific Applications (BTW)*, pages 107–126, 2001.

[KKKK02a] M. Keidl, A. Kemper, D. Kossmann, and A. Kreutz. Verteilte Metadatenverwaltung und Anfragebearbeitung für Internet-Datenquellen. *Informatik Forschung und Entwicklung*, 17(3):123–134, 2002.

[KKKK02b] M. Keidl, A. Kreutz, A. Kemper, and D. Kossmann. A Publish & Subscribe Architecture for Distributed Metadata Management. In *Proc. of the 18th Intl. Conference on Data Engineering (ICDE)*, pages 309–320, 2002.

[KSK02] M. Keidl, S. Seltzsam, and A. Kemper. Flexible and Reliable Web Service Execution. In *Proc. of the 1st Workshop on Entwicklung von Anwendungen auf der Basis der XML Web-Service Technologie*, pages 17–30, 2002.

[KSSK02] M. Keidl, S. Seltzsam, K. Stocker, and A. Kemper. ServiceGlobe: Distributing E-Services across the Internet (Demonstration). In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 1047–1050, 2002.

[Ley01] F. Leymann. Web Services Flow Language (WSFL 1.0). `http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`, 2001. IBM.

[Mar99] H. Marais. Compaq's Web Language. `http://www.research.compaq.com/SRC/WebL/WebL.pdf`, 1999. Compaq.

[NET] Microsoft .NET. `http://www.microsoft.com/net`.

[Pas] Microsoft Passport. `http://www.passport.com`.

[Pro] Liberty Alliance Project. `http://www.projectliberty.org`.

[RV02] E. Rahm and G. Vossen, editors. *Web & Datenbanken: Konzepte, Architekturen, Anwendungen*. dpunkt-Verlag, 2002.

[SBK01] S. Seltzsam, S. Börzsönyi, and A. Kemper. Security for Distributed E-Service Composition. In *Proc. of the 2nd Intl. Workshop on Technologies for E-Services (TES)*, volume 2193 of *Lecture Notes in Computer Science (LNCS)*, pages 147–162, 2001.

[Sun] Sun Open Net Environment (Sun ONE). `http://www.sun.com/sunone`.

[Tha01] S. Thatte. XLANG: Web Services for Business Process Design. `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`, 2001. Microsoft.

[UDD00] Universal Description, Discovery and Integration (UDDI) Technical White Paper. `http://www.uddi.org`, 2000. White Paper.

[WSP] HP Web Services Platform. `http://www.hp.com/go/webservices`.

Efficient Assembly of Product Structures in Worldwide Distributed Client/Server Environments

E. Müller, P. Dadam
University of Ulm
Faculty of Computer Science
Dept. Databases and Information Systems
{mueller,dadam@informatik.uni-ulm.de}

M. Feltes
DaimlerChrysler
Research and Technology
Dept. RIC/ED, Ulm
{michael.feltes@daimlerchrysler.com}

Abstract: The efficient management of product-related data is a big challenge for many manufacturing companies, especially the larger ones like DaimlerChrysler. So-called Product Data Management (PDM) systems are used to tackle this task. But especially in worldwide distributed product development environments, where low bandwidth networks are used, response times of PDM systems often raise to several minutes even for simple user actions. The main reason for this is the amount of wide area network communication caused by poor implementations of PDM systems which a) often use the underlying database management system like a stupid record manager, and b) do hardly know anything about the distribution of the data. In this paper we introduce an extension of directed acyclic graphs which enables us to adequately represent product structures. On the basis of this representation we show how some additional information describing the data distribution can be used to assemble distributed product structures very efficiently.

1 Introduction

Product development is a time-consuming and costly process. Keen competition especially forces the manufacturing companies to shorten this process in order to survive. Besides the wellknown CAD and CAE tools, which help to optimize the completion of *intra*-disciplinary tasks, so called Product Data Management (PDM) systems have been established during the last years in order to support the *inter*-disciplinary tasks at best (cf. [MOn], [MP], [WT], [CIM97]).

PDM systems claim to be the information backbone serving all users involved in the development process (cf. [OLR95]). This means that all data relevant to somebody in this process have to be managed by the PDM system – beginning at the core product structure which describes the composition hierarchy of parts and subparts, and ending at all describing documents like specifications, CAD files, work orders, simulation results, and so on (cf. figure 1). Especially in case of complex products like cars, PDM systems have to manage thousands of such objects and provide these data to thousands of users – with acceptable response times!

It is not only the amount of data which makes a product complex. It is also the fact that products can be customized or *configured* and therefore typically exist in several different

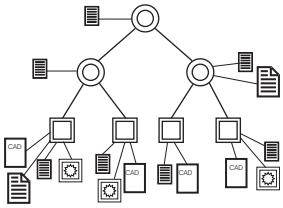


Figure 1: A simplified product structure with several document types

versions: A car for example may be offered in a "basic model" which can be configured according to the customers' demands by equipping it with lots of additional features, e. g. a sunroof, a navigation system, xenon headlights and so on. Easy to imagine that the number of different producible cars can be extremely high, but the difference between two configurations may be rather small as all configurations share a lot of common parts. So instead of storing each configuration separately (which would lead to totally inefficient structure handling) the structures of all supposable product instances are combined in one large structure. To retrieve the structure of a certain product instance, parts of this structure may be ruled out by appropriate conditions.

A typical way of using PDM systems is to *navigate* in the product structure according to some configuration options (like the features in the car example). Users begin at the top-level element of the product structure (i. e. the product itself) and expand the subjacent level of the structure. This so-called *single-level expand* is repeated until the user finds what he looks for. In doing so, today's PDM systems – for various reasons they typically sit on top of relational database management systems – use SQL as a simple record manager: The navigational traversal of the product tree is typically translated nearly one-to-one into single isolated SQL queries. This stepwise navigation also works for the *multi-level expand* which expands the entire product structure by recursively applying the single-level expand method.

A closer look to the representation of product structures within the database shows how these expand actions work: Typically parts consist of several subparts (which are parts as well), and sometimes one subpart may be integrated into several parts. This recursive n:m-relationship is realized by so-called link-objects. In distributed environments the link-objects also identify the databases where the related parts are stored. In today's PDM systems an expand action for a certain part first retrieves all link-objects that identify the corresponding subparts. In a second step the subparts are retrieved from the database(s) indicated in the link-objects.

This *naive (one-object-at-a-time* or *one-level-at-a-time)* approach leads to a large number of SQL queries (and result messages, of course). In environments, where the DBMS and PDM system are connected via high-speed netwoks with low latency times, this may not cause too much harm. The picture may change dramatically, however, if the users are working in geographically distributed environments. Some experiments in prototypical but realistic PDM environments at DaimlerChrysler have shown that response times may rise by orders of magnitude, e. g. from 1–2 minutes in the local context to 30 minutes in

the "intercontinental" context. The reasons for these inacceptable response times are the typically long latency times and low bandwidth of wide area networks.

Obviously, the aim must be to cut down response times of expansion operations by a solution that a) minimizes the number of communications (i. e. the number of SQL queries) between distributed locations, and b) in doing so, does not increase the volume of transmitted data. A very promising way to achieve this goal is to provide a function shipping approach instead of today's stepwise data shipping.

In [MDEF01] we describe a first approach for pushing the navigational process from the PDM system to the DBMS. Substituting several isolated consecutive SQL queries by one recursive SQL query (cf. [ANS99], [EM99], [IBM01]) may dramatically speed up the expand methods in environments having only one central data server and several distributed PDM servers. However, if the data is stored distributedly – i. e. the tables storing parts and link-objects are split into several partitions which are distributed across several database servers – there is no way to efficiently assemble the entire product structure by one single recursive query: At query generation time there is no information available about the partitions that have to be accessed for assembling the given part. This information can only be obtained by interpreting link-objects. As SQL does not provide the ability to dynamically switch to different data sources during query execution according to such "run-time interpreted" data, the recursive query either considers a) only one partition or b) the union of all partitions. In the first case one needs to generate a sequence of recursive queries each of which only collects the parts and link-objects stored in one partition. In the latter case too many tuples, which do not have any effect on the resulting structure, are transmitted to the executing server.

In principle, the first approach is the basis of the solution presented in this paper. We provide additional information about which parts at which servers have to be retrieved when assembling the product structure. This index-like information, the so-called *object link and location catalog*, enables us to contact each server involved in a product structure assembly only once. It avoids multiple retrieval of multiply occurring parts within the structure, it allows queries to different database servers to be executed in parallel, and it helps to minimize the number of communications between database servers.

The rest of this paper is organized as follows: Section 2 introduces an extension of a common directed acyclic graph, the so-called *directed acyclic condition graph*. This enables us to adequately represent configurable product structures. In section 3 we define the object link and location catalog. Usage and benefits of this catalog are described in section 4. The algorithms for handling the catalog are presented in section 5. Related work is discussed in section 6. Section 7 concludes and gives an outlook to further work.

2 Product Structure in Terms of a DAG

2.1 Representing Product Structures

At first glance a product structure can be adequately represented by a simple directed acyclic graph (DAG, cf. [CLR96],[Sed88]): Each project, part and subpart is a vertex of such a graph, and any directed relation between them – e.g. the "part-uses-subpart" relation – is a corresponding edge. But unfortunately, the simple DAG lacks the ability to express the flexibility of a product structure regarding product configuration and structure options (confirm section 1). It would be neccessary either to store each product instance in an isolated DAG (which is impossible for products with a large number of possible configurations like cars) or to simply "merge" the structures of all producible product instances losing the capability to correctly extract the structure of a certain product instance. In order to solve this problem we need the ability to express whether a certain edge of the DAG is a member of an actual corresponding product instance or not. This can be achieved by extending the DAG by appropriately defined configuration conditions.

2.2 Extending DAGs for Product Structure Requirements

We assume $\mathcal{O}$ to be a set of available options $\mathcal{O} = \{\chi_1, \chi_2, \ldots, \chi_n\}$ that can be used to configure product instances. A user can choose some, all or none of these options in order to indicate which optional parts have to be included into the basic product structure. We describe this situation by the following formalism in the style of propositional calculus (cf. [Fit90], [LE78]):

$\mathcal{O}$ can be viewed as a set of atomic formulas. When the user chooses some options, he defines an *assignment* $\bar{\mathcal{A}} : \mathcal{O} \to \{0, 1\}$, i. e. selected options are assigned the value 1, the others are set to 0.

Dependent on $\mathcal{O}$ we introduce the set $\mathcal{C}$ of logic expressions – called *conditions* – which are solely built over the atomic formulas in $\mathcal{O}$, i. e. the elements of $\mathcal{O}$ can be combined with the operators AND ($\wedge$), OR ($\vee$) and NOT ($\neg$). For completeness reasons we assume the "true condition" $\top$ (always true) to be an element of $\mathcal{C}$, too. Now we extend $\bar{\mathcal{A}}$ to $\mathcal{A} : \mathcal{C} \to \{0, 1\}$ which indicates whether an expression $c \in \mathcal{C}$ evaluates to true (i. e. 1) or not.

Definition 1: Directed Acyclic Condition Graph (DACG)

> A *Directed Acyclic Condition Graph* G^c is an extended DAG defined as follows:
>
> $G^c = (V, E, \mathcal{C})$ where $E \subseteq V \times V \times \mathcal{C}$, V is the set of vertices, $\mathcal{C}$ is a set of conditions, and E is the set of condition-annotated edges between vertices.

The semantics is as follows: An edge from vertex u to vertex v is only valid, if the an-

notated condition c evaluates to TRUE (i. e. $\mathcal{A}(c) = 1$) for a given assignment $\bar{\mathcal{A}}$.[1] In general we denote this edge $u \xrightarrow{c} v$, but if $c = \top$ we omit the condition and write simply $u \rightharpoonup v$. Similarly, a path from vertex u to vertex v in a DACG G^c is written as $\langle u \xrightarrow{c_1} n_1 \xrightarrow{c_2} \ldots \xrightarrow{c_k} n_k \xrightarrow{c_{k+1}} v \rangle$.

For our investigations in subsection 3.2 we introduce the transitive closure of a DACG here:

Definition 2: Transitive closure of a DACG

The transitive closure G^{c*} of a DACG $G^c = (V, E, \mathcal{C})$ is defined as follows:

$G^{c*} = (V, E^*, \mathcal{C}^*)$ where $E^* = E \cup \{(u \xrightarrow{c} v) | u, v \in V \wedge$
$\exists d = \langle u \xrightarrow{c_1} n_1 \xrightarrow{c_2} \ldots \xrightarrow{c_k} n_k \xrightarrow{c_{k+1}} v \rangle$ in $G^c, (k > 0)$,
and $c = c_1 \wedge c_2 \wedge \ldots \wedge c_{k+1}\}$ and $\mathcal{C}^* = \mathcal{C} \cup \{c | u \xrightarrow{c} v \in E^*\}$.

Like in ordinary DAGs an edge of a transitive closure of a DACG connects vertices that are starting and ending points of a path in the original graph. In case of DACGs each edge of the transitive closure is annotated with the conditions which are related to the edges of the corresponding path, connected by the "AND" operator.

2.3 Partitioning DACGs

In typical scenarios product structures are distributed accross several partners and suppliers. In such environments it is realistic to assume that each involved partner stores only those portions of the structure which he is responsible for or which correspond to parts he produces. So, looking at the entire product structure as a DACG, each server manages some sort of *partition* of this DACG.

We define a partition P_{s_i} of a DACG as follows:

Definition 3: Partition of a DACG

Assume $G^c = (V, E, \mathcal{C})$ a DACG and $f : V \to S$ a function that "marks" all the vertices of G^c with an element of S. For each element $s \in S$ there exists one subgraph P_s of G^c where

$$P_s = (V_s, E_s, \mathcal{C}) | V_s = \{v \in V | f(v) = s\} \wedge$$
$$E_s = \{(u \xrightarrow{c} v) \in E | u \in V_s \vee v \in V_s\}$$

The *connected components*[2] of each P_s are called *partitions* $P_{s_i} = (V_{s_i}, E_{s_i}, \mathcal{C})$ of G^c.

[1] In the automotive industry the condition c for example may guarantee that "v is only part of u if the car in scope has to be equipped with a sunroof".

[2] The *connected components* of a directed graph $G = (V, E)$ correspond to the connnected components of the undirected version $G' = (V, E')$ of G. In the same sense we use the term for a DACG, too.

Remark: a partition P_{s_i} does not represent a DACG as defined in definition 1, since E_{s_i} may contain at least one edge $(u \xrightarrow{c} v)$ where either $u \notin V_{s_i}$ or $v \notin V_{s_i}$! These edges combine two differently marked partitions and belong to both connected partitions!

In our environment S would be the set of servers or PDM systems participating in a product development environment. The function f would map each elementary object (e. g. an assembly) to the server it is stored at.

Figure 2 shows an example of a very simple DACG split into several partitions. In this example, the set of marks is S=$\{A, B, C, D\}$, and the mapping function f assigns the following values: $f(1)$=$f(2)$=$f(3)$=$f(13)$=A, $f(4)$=$f(12)$=B, $f(5)$=$f(6)$=$f(8)$=$f(9)$=$f(10)$=C, $f(7)$= $f(11)$=D. The set of conditions is $\mathcal{C}$=$\{c_1, c_2, c_3, c_4, \top\}$. The partitions of G^c are marked with grey shapes.

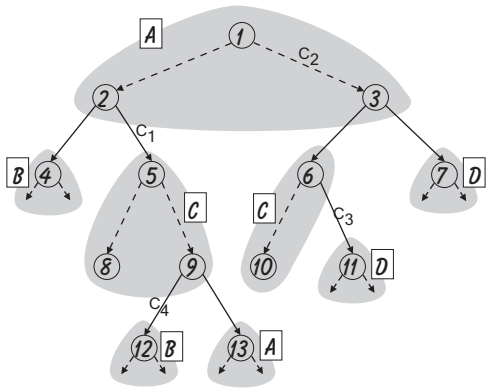


Figure 2: A simple distributed DACG G^c

3 Object Link and Location Catalogs

3.1 Conventional Assembly Strategies

Partitioning as introduced in subsection 2.3 enables us to specify the distribution of a DACG across several servers. In order to reconstruct a suchlike distributed DACG – or, speaking in terms of product structures, to assemble an entire product of all parts and subparts – there exist several well-known, conventional strategies (cf. [MGS+94]):

1. Starting with the top-level element (the root node of the graph), fetch recursively all directly related subparts of the parts already retrieved (*one-level-at-a-time*). – As already outlined, this is the naive, very inefficient approach.
2. Ship all remote data to the location where the top-level element is stored (or where the action is started) and assemble the entire complex object subsequently at that

site. – This strategy may be very expensive: The resulting complex object may consist of only a very small fraction of all transmitted vertices (i. e. simple objects) due to some user-selected conditions. Unfortunately, the objects not included in the result possibly caused a non-negligible transmission overhead.

3. The server starting the action becomes the "master" of the activity (storing a partition we call P_{master}), all other servers behave like "slaves" (storing $P_{s_1}, \ldots, P_{s_n}$). The master assembles the desired structure locally (as far as possible) and requests missing parts of the structure from the slaves all times it reaches an edge that combines P_{master} with one of the other P_{s_i}. As long as the result of a remote server s_i contains an edge that connects P_{s_i} to P_{s_j}, the master has to request the missing part of the structure from the server storing P_{s_j}.

   To illustrate this approach, we assume S (cf. figure 2) to be the set of participating product data servers. Server A at first retrieves vertices $1, 2$, and 3, interprets the edges $2 \rightharpoonup 4$, $2 \rightharpoonup 5$, $3 \rightharpoonup 6$, and $3 \rightharpoonup 7$ (to simplify matters we omit conditions here) successively and sends corresponding requests to servers B ("assemble substructure with root 4"), C ("assemble substructure with root 5", "assemble substructure with root 6"), and D ("assemble substructure with root 7). The result of server B does not contain any edges that combine two partitions, but the results of server C do: They contain $9 \rightharpoonup 12$ and $9 \rightharpoonup 13$, and $6 \rightharpoonup 11$ respectively, so the master (i. e. server A) has to request the substructures with roots 12, 13 and 11 from the corresponding servers B, A(!), and D. After this the structure is complete.

   There is one major disadvantage with this approach: Each server may be contacted more than once within *one* activity as it happend to the servers B, C, and D in our example. This causes superfluous net traffic and therefore – especially in wide area networks – leads to long response times.

4. Instead of having only *one* master which requests substructures from remote locations, each server is allowed to do so. In our example server A requests the substructures with roots 5 and 6 from server C which in turn requests the substructure with root 12 from server B, 11 from server D etc. and returns the combined results to A. A finishes the action after receiving the responses of all servers according to their requests.

   This approach has the disadvantage mentioned in approach 3, too. In addition to this, if there exist cyclic dependencies within the servers storing partitions of the DACG (e. g. server A references server C which in turn references A) parts of the graph may be sent through the network multiple times (A sends to C, which returns the combined result back to A). This is absolutely dissatisfying!

5. One could also use a mixture of approaches 3 and 4. A server, for example, which is reachable only through a very slow network connection may not be allowed to request subtrees from remote servers, except it is the master according to approach 3. – This may eliminate some low speed network communications, but the disadvantages cannot be obviated really.

As a matter of fact, using conventional algorithms there is no way to reconstruct a widely distributed DACG causing only minimal communication effort. Obviously, the reason for poor performance is that – depending on the distribution of the DACG – remote servers may have to be contacted multiple times within one single assembly operation. So, what we need is some additional information about the distribution of the DACG to avoid such multiple server connections when accessing more than one partition of the structure. In our example server A initially does not know anything about the edge between vertices 9 and 12, so A does not know that there exists an additional (transitive) dependency from A to B. However, if A knew this, it could request vertex 12 from B directly, saving one communication to B if the requests for vertices 4 and 12 were combined! So what we need is some sort of a "virtual edge" e. g. between vertices 2 and 12. Suchlike edges will be stored in the object link and location (OLL) catalog.

3.2 Definition of OLL Catalog

The challenge is to construct the OLL catalog in that way that it is a) minimal in size (that means it does not contain irrelevant or obsolete information) and b) complete (no information neccessary is missing). Based on the DACG introduced in subsection 2.2 we define the OLL catalog as follows:

Definition 4: OLL Catalog

> Assume $G^c = (V, E, \mathcal{C})$ to be a DACG, and $f : V \rightarrow S$ a function for marking vertices. Assume $G^{c*} = (V, E^*, \mathcal{C}^*)$ to be the transitive closure of G^c. Then the *OLL catalog* E' of G^c is defined as:
>
> $E' = \{u \xrightarrow{c} v \in E^* \setminus E | \exists d = \langle u \xrightarrow{c_1} n_1 \xrightarrow{c_2} \ldots \xrightarrow{c_k} n_k \xrightarrow{c_{k+1}} v \rangle \text{ in } G^c,$
> $(k > 0), \text{where} f(n_i) \neq f(u) \wedge f(n_i) \neq f(v), 1 \leq i \leq k\}.$

In other words: The OLL catalog contains a virtual connection of two vertices u and v if v is reachable (in the original DAG) from u by only traversing vertices that are marked differently from u and v. Figure 3 shows the DACG of figure 2 extended by edges of its corresponding OLL catalog ($2 \xrightarrow{c_1 \wedge c_4} 12$, $2 \xrightarrow{c_1} 13$, and $3 \xrightarrow{c_3} 11$).

4 Using OLL Catalogs

4.1 Description of the Approach

The goal of the OLL catalog is to enable performing of recursive expansions a) in one single path and b) to visit each server involved only once. This is achieved by providing enough information in the OLL catalog such that all "transitive" remote requests can be performed by the initiating server. If, for example, the product structure in figure 3 has to

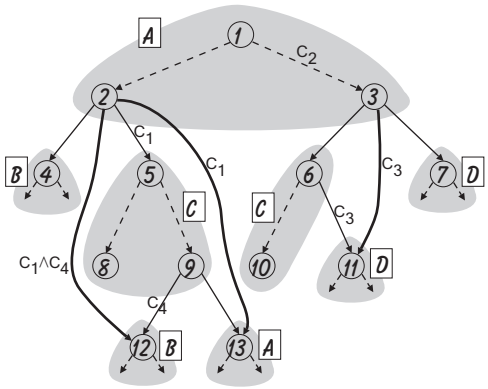


Figure 3: The DACG G^c of figure 2 extended by the edges of the OLL catalog

be expanded by server A, server A will contact servers C, B, and D in order to retrieve the vertices stored there, while servers B, C, and D only perform local expansions.

The multi-level expand of a product structure can be split into three steps: First, upon request the server storing the top-level element of the structure expands the structure according to the locally available structure information. During this recursive descent the server collects unresolved references to remote parts. In the second step, recursive queries are generated for retrieving the remote parts. Third, the queries are executed at the remote sites in parallel and return partially assembled substructures. We will show this procedure by performing a multi-level expand of vertex 1 in the example in figure 3.

We assume c_1=c_2=c_3=c_4=1, i. e. all partitions of the DACG have to be retrieved.

At first, server A expands the structure of vertex 1 locally: The edges pointing from vertex 1 to vertices 2 and 3 are retrieved and interpreted. As vertices 2 and 3 are stored locally, they can be fetched at once. As there are no edges in the OLL catalog starting at vertex 1, server A immediately tries to expand the structure of vertex 2. The corresponding edges are fetched from the local partition of server A, but they point to the vertices 4 and 5 at servers B and C respectively. Instead of retrieving the two remote vertices immediately, server A collects them in a set of unresolved references for later usage.

At this point server A accesses the OLL catalog again: There exist two entries in the catalog, pointing from vertex 2 to vertices 12 and 13 at servers B and A respectively. The conditions attachted to both vertices evaluate to true ($c_1 \wedge c_4$=1, and c_1=1, too), so both vertices are in the scope of the result. OLL catalog entries pointing to a partition that is stored locally are processed immediately, the target vertices of the remaining entries are added to the set of unresolved references. This causes all locally stored partitions, which are reachable from the vertex in scope (in the example this is vertex 1), to be expanded in one recursive descent. So, server A now proceeds by expanding vertex 13 according to the method just described for vertex 2.

After finishing the assembly of vertex 13, server A turns towards vertex 3. All edges in the

DACG as well as in the OLL catalog, which start at vertex 3, point to vertices at remote partitions. Therefore they are collected in the set of unresolved references, too.

At this moment, server A finishes its local recursion. The set of unresolved references contains the following vertices: 4, 12, 5, 6, 11, and 7.

Now the remote servers can be instructed to locally expand the missing parts of the structure: Server B has to assemble the subgraphs of vertices 4 and 12, server C the subgraphs of vertices 5 and 6, and server D the subgraphs of vertices 11 and 7. Note that each server needs to be contacted only once in order to initiate the local expands!

As each of the three remote servers B, C, and D performs the same steps we show the local expansion at server C only. Similar to the procedure already described for server A, the server C tries to expand vertex 5 (vertex 6 either is processed in parallel or after finishing the expansion of vertex 5) using the edges in the local DACG which references vertices 8 and 9. Vertex 8 is a leaf in the graph and therefore cannot be further expanded. In contrast, vertex 9 has references to the vertices 12 and 13 at servers B and A respectively. As vertex 13 is already expanded by server A, and vertex 12 is processed by server B in parallel, the recursive expansion stops here. Finally, the result is returned to server A.

After receiving all subgraphs from the involved remote servers, the entire structure is available at server A.

Further details can be found in the algorithms in section 5.

4.2 Benefit of Using OLL Catalogs

Using the naive *one-object-at-a-time* approach, the expansion of a structure needs as many remote accesses as remote nodes are involved in the structure. In figure 3, if the partitions stored on servers B and D only contain the vertices 4, 12, 7, 11 respectively, then nine remote communications are necessary for the expansion.

The approach using recursive queries without OLL catalogs will require as many remote accesses as remote partitions are involved in the expansion of the structure. In our example this still would lead to six remote communications.

By the usage of OLL catalogs the number of remote communications can be reduced to the number of remote servers participating in the expansion. As each server in our example stores two partitions of the structure, half of the six remote accesses can be saved. Furthermore, in contrast to the other approaches, these remaining three requests can be processed in parallel. Thus the overall expand does not take much longer than the most time-consuming remote expand of a single substructure! If we assume the three remote servers B, C, and D of our example to require similar execution times, the response time can be cut down by approximately another two thirds!

In general, the usage of OLL catalogs reduces the amount of remote accesses to the lowest possible extent, namely the number of database servers storing product data. Obviously the benefit of this approach depends on the distribution of data. If there are only very few servers storing more than one partition of the structure, it may not lead to much less

communications. In heavily distributed environments, however, OLL catalogs may save half of the remaining remote communications and even more.

Nevertheless, if the number of communications can not be cut down by the OLL catalog due to a "smooth" data distribution, response times may still dramatically decrease: The recursive queries for assembling substructures on different servers can be executed in parallel since the transitive dependencies between different partitions are resolved by the OLL catalog and already considered when initiating the remote queries.

So, for all cases of distribution, OLL catalogs allow efficient assembly of product structures. Improvements regarding the response times are achieved by reducing the number of communications and by parallel execution of remote queries.

5 Algorithms for Handling OLL Catalogs

The algorithms in this section work on a DACG $G^c = (V, E, \mathcal{C})$. We use some notations we informally introduce here:

S is the set of servers (hosts) which store one or more partitions of G^c.

$f(x)$ is a function $f : V \rightarrow S$.

$z = \langle a, b, c \rangle$ denotes a triple z consisting of the elements a, b, and c. The elements can be accessed via $z.a$, $z.b$ and so on.

$g^{(f(v))}(x)$ means that the function $g(x)$ has to be evaluated at the server storing vertex v.

$s[z.x \leftarrow y]$ denotes an update action on all tuples z within the set s. The value of attribute x is set to y.

OLL denotes the OLL catalog at the current (working) server (i. e. the server that runs the procedure accessing the OLL catalog).

$\mathcal{A}$ describes the structure options in scope (as introduced in subsection 2.2).

5.1 OLL Catalog Initialization

The initialization process of the OLL catalog is distributed across all servers which store parts of the structure. In order to create catalog entries we walk through the entire structure depth-first, starting at the server that stores the root of the structure. At each vertex we test if a OLL edge has to be created to one of the already visited vertices in the path down from the root vertex (cf. definition 4).

The initialization procedure is initially called with `Init(root,`ε`,`$\emptyset$`)`.

Algorithm 1: (Initialization of the OLL catalog)

Init(in vertex u, in condition precond, in borders B)

1 *new_edges* $\leftarrow \emptyset$

```
2   sorted_B ← ∅
3   if (B == ∅) then seq ← 1
4   else
5       sorted_B ← sort(B) on B[z.seq_number] desc
6       seq ← max(B[z.seq_number]) + 1
7   endif
8   if (precond == ε) then
9       for all ⟨v, c, q⟩ ∈ sorted_B do
10          if (v -c-> u ∉ E) then
11              OLL ← OLL ∪ {v -c-> u}
12              if (f(u) == f(v)) then
13                  break
14              else
15                  new_edges ← new_edges ∪ {v -c-> u}
16              endif
17          endif
18      enddo
19  endif
20  for all u -c-> p ∈ E do
21      succ.push ⟨u -c-> p, precond⟩
22  enddo
23  while (succ.test != NULL) do
24      ⟨u -c-> v, cond⟩ ← succ.pop
25      if (f(u) == f(v)) then
26          for all v -c̄-> w ∈ E do
27              succ.push ⟨v -c̄-> w, cond ∧ c⟩
28          enddo
29      else
30          B' ← (B[z.cond ← z.cond ∧ cond ∧ c]\
31              {b ∈ B : f(b) == f(u)}) ∪ {⟨u, c, seq⟩}
32          succ_edges ← Init^(f(v))(v, ε, B')
33          for all x -c-> y ∈ succ_edges do
34              if (x == u) then
35                  OLL ← OLL ∪ {x -c-> y}
36              else
37                  new_edges ← new_edges ∪ {x -c-> y}
38              endif
39          enddo
40      endif
41  enddo
42  return new_edges
```

Method: The tree traversal is started at the root vertex. Every time a link $u \xrightarrow{c} v$ from a partition P_τ to an other partition P_μ is traversed, we include u (and the preconditions c' which define how to reach u) in a set B of consecutively numbered "border objects" (these are vertices which have at least one edge to a vertex that belongs to a different partition). If B already contains a border object $b \in B | f(b) = f(u)$ – this is true if the path from

the root of G^c down to u contains elements which are stored at the same server as u but in partitions different from P_τ – we substitute b with u. After that the server hosting P_μ is instructed to initialize its OLL catalog concerning P_μ regarding B. Here the first step is to create OLL catalog entries, i. e. we loop over $b \in B$ descendingly ordered by their sequence numbers, and create links from these b to v with respect to the precondition of b.[3] This loop stops if either all border objects are processed or a border object in scope is stored at the same server as P_μ. Thereafter the algorithm proceeds as described for the root server.

If a server finishes its OLL catalog initialization, it returns the created edges to its caller. This server copies all edges $u \xrightarrow{c} v$ where u is stored locally to its own OLL catalog. The remaining edges are added to the set of self-created OLL catalog entries and returned to its caller in turn. The procedure stops when the root server finishes.

5.2 OLL Catalog Extension

Product structures are frequently updated during the product development process. In most cases they do not affect the distribution of data. Object migration occurs rather seldom, as typically all partners and suppliers are interested in managing the data concerning their work share locally. Sometimes, however, product structures are not completely defined at startup of a project, so additional suppliers may have to be integrated into the development framework later. In this case updates of the OLL catalog have to be performed.

Algorithm 2 shows a fragment of a structure update procedure that calls the Update-procedure (algorithm 3) regarding the servers the newly connected objects are stored at.

Algorithm 2: (Extension of product structure)

CreateNewEdge(in vertex u, in vertex v, in condition c)

```
1   ...
2   if (f(u) == f(v)) then
3       dset ← Update^(f(u))(u, ⟨v, c⟩, c, ∅)
4   else
5       dset ← Update^(f(u))(u, ⟨v, ε⟩, c, {⟨u, c, 0⟩})
6   endif
7   ...
```

Algorithm 3: (Extension of OLL Catalog)

Update(in vertex x, in ⟨vertex u, condition precond⟩,
in condition $\bar{c}$, in borders B)

```
1   new_edges ← ∅
```

[3] The border object u has not to be regarded since $u \xrightarrow{c} v \in E$ (storing this edge in the OLL catalog would not lead to any additional information).

```
2   if (B == ∅) then
3       seq ← 0
4   else
5       seq ← min(B[z.seq_number]) − 1
6   endif
7   if (∃ w −c→ x ∈ E) then
8       for all w −c→ x ∈ E do
9           pred.push ⟨w −c→ x, c̄⟩
10      enddo
11  else
12      pred_edges ← Init^(f(u))(u, precond, B)
13      for all u −c'→ v ∈ pred_edges do
14          if (f(u) == f(x)) then
15              OLL ← OLL ∪ {u −c'→ v}
16          else
17              new_edges ← new_edges ∪ {u −c'→ v}
18          endif
19      enddo
20  endif
21  while (pred.test ! = NULL) do
22      ⟨x −c→ y, cond⟩ ← pred.pop
23      if (f(x) == f(y)) then
24          if (∃ w −c''→ x ∈ E) then
25              for all w −c''→ x ∈ E do
26                  pred.push ⟨w −c''→ x, cond ∧ c⟩
27                  if (seq == 0) then
28                      precond ← precond ∧ c
29                  endif
30              enddo
31          else
32              pred_edges ← Init^(f(u))(u, precond, B)
33              for all u −c→ v ∈ pred_edges do
34                  if (f(u) == f(x)) then
35                      OLL ← OLL ∪ {u −c→ v}
36                  else
37                      new_edges ← new_edges ∪ {u −c→ v}
38                  endif
39              enddo
40          endif
41      else
42          if (¬∃b ∈ B|f(x) == f(b)) then
43              pred_edges ← Update^(f(x))(x, ⟨u, precond⟩,
44                      cond ∧ c, B ∪ {⟨x, cond ∧ c, seq⟩})
45          else
46              pred_edges ← Update^(f(x))(x, ⟨u, precond⟩,
47                      cond ∧ c, B)
```

```
48              endif
49              for all u -c-> v ∈ pred_edges do
50                  if (f(u) == f(y)) then
51                      OLL ← OLL ∪ {u -c-> v}
52                  else
53                      new_edges ← new_edges ∪ {u -c-> v}
54                  endif
55              enddo
56          endif
57      enddo
58      return new_edges
```

Method: The general idea behind algorithm 3 is as follows:
Assume $u \xrightarrow{c} v$ to be the new edge. If we knew about all border objects in the path from the root object down to u we could simply call the Init-procedure (cf. algorithm 1) for u. To obtain information about all relevant border objects is the main task of algorithm 3. The procedure is to go backwards in the structure from u up to the root object and to collect all border objects and the conditions along the path. When the root object is reached the Init-procedure can be called.

Returning from the recursive descend each server includes the relevant new OLL edges – created by Init – in its local OLL catalog, the remaining edges are returned to the calling server.

5.3 OLL Catalog Usage

When expanding (or assembling) product structures, the usage of OLL catalogs is rather simple. We only have to distinguish between the "master" server (i. e. the server that stores the root vertex of the requested structure) and the other ones, the "slaves". The client which requests the structure initiates the multi-level expand by calling XMultiLevelExpand (cf. algorithm 4) which in turn calls the masterMLE and slaveMLE procedures (algorithms 5 and 6 respectively).

Algorithm 4: (Product structure expansion)

```
XMultiLevelExpand(in node u, in assignment A)
1   subgraph ← (∅, ∅)
2   remotenodes ← ∅
3   masterMLE^(f(u))(u,A,subgraph,remotenodes)
4   for all s ∈ S in parallel do
5       R_s = {v ∈ remotenodes : f(v) = s}
6       subgraph_s ← (∅, ∅)
7       if (¬(R_s == ∅)) then
8           slaveMLE^(s)(R_s, A, subgraph_s)
9           Merge(subgraph, subgraph_s)
```

```
10          endif
11      enddo
12      return subgraph
```

Method: XMultiLevelExpand is called with the vertex u to expand, and the assignment $\mathcal{A}$ of structure options the user selected. First the local structure is expanded (masterMLE), resulting in (1) a preliminary subgraph and (2) all nodes on remote servers which have to be expanded subsequently. The subtrees of all remote servers are requested in parallel (cf. lines 4 – 11). The Merge procedure (line 9) simply creates the union of all resulting subgraphs (i. e. $V = \bigcup_{s \in S} \tilde{V}_s, E = \bigcup_{s \in S} \tilde{E}_s$ where $(\tilde{V}_s, \tilde{E}_s)$ is the resulting subgraph of server s).

Algorithm 5: (Expansion according to the OLL catalog (master))

```
masterMLE(in node u, in assignment A,
          out (Ṽ_s, Ẽ_s), out nodes remotenodes)
1   Ṽ_s ← {u}
2   Ẽ_s ← ∅
3   successors ← {u}
4   for all x ∈ successors do
5       successors ← successors \ {x}
6       for all x -c-> y ∈ E ∪ OLL|A(c) do
7           if x -c-> y ∈ E then
8               Ẽ_s ← Ẽ_s ∪ {x -c-> y}
9           endif
10          if (f(x) == f(y)) then
11              Ṽ_s ← Ṽ_s ∪ {y}
12              if (¬marked_visited(y)) then
13                  successors ← successors ∪ {y}
14              endif
15          else
16              remotenodes ← remotenodes ∪ {y}
17          endif
18      enddo
19      mark_visited(x)
20  enddo
21  return
```

Method: masterMLE is called to assemble vertex u considering the assignment $\mathcal{A}$. The two nested loops in line 4 and 6 traverse the local substructure. In lines 7 – 9 the local edges are collected (OLL edges are not included in the result!). Local vertices are then added to the result, remote nodes are collected separately (lines 10 – 17). Visited vertices are marked in order to avoid multiple processing (line 19). The result of masterMLE consists of a) the local subgraph and b) a set of all directly or indirectly referenced vertices stored at remote sites.

Algorithm 6: (local expansion of a PS according to local OLL catalog entries (slave))

slaveMLE(in R_s, in assignment $\mathcal{A}$, out ($\tilde{V_s}, \tilde{E_s}$))

```
1   Ṽ_s ← R_s
2   Ẽ_s ← ∅
3   successors ← R_s
4   for all x ∈ successors do
5       successors ← successors \ {x}
6       for all x -c-> y ∈ E ∪ OLL|A(c) do
7           if x -c-> y ∈ E then
8               Ẽ_s ← Ẽ_s ∪ {x -c-> y}
9           endif
10          if (f(x) == f(y)) then
11              Ṽ_s ← Ṽ_s ∪ {y}
12              if (¬marked_visited(y)) then
13                  successors ← successors ∪ {y}
14              endif
15          endif
16      enddo
17      mark_visited(x)
18  enddo
19  return
```

Method: In contrast to the masterMLE procedure, slaveMLE is called with a *set* of vertices which are the root objects of local substructures. The rest of slaveMLE works similar to masterMLE, except that no remote objects have to be collected.

6 Discussion

During the past few years a lot of research work has been performed in the context of distributed query processing in all kinds of databases (cf. [Gra93], [HMS92], [Kos00], and [YM98]). The very special but important aspect of assembling distributed objects, however, has received little attention.

In [KGM91] and [MGS+94] an "assembly" operator is introduced for combining distributed complex objects causing only a small number of communications to remote sites by doing as much work as possible at each location which is involved in the user's request: The remote sites assemble the object status locally as far as possible and return partial objects which have to be checked for further unresolved references. If any, the missing parts are retrieved from the corresponding servers. This is similar to the approach described in section 1, where a sequence of recursive SQL queries is executed to assemble the entire product structure. The problems are similar as well: Remote sites may be contacted more than once, and parts occurring more than once within the structure are retrieved multiply.

An evaluation strategy for executing recursive queries in distributed deductive databases is described in [NCW93]. The paper focuses on minimizing the distribution costs based on

the idea of semi-joins in conventional databases. Avoiding multiple calls to a remote site is not in the scope of the approach presented there.

Initialization and usage of OLL catalogs are closely related to distributed or parallel execution strategies for transitive closure (cf. [CCH93], [HAC90]). The distributed product structure in our scenario may be seen as some kind of *semantic fragmentation* of the data according to the workshare defined by collaborating partners and suppliers. [HAC90] describes an approach similar to the one presented here: First, fragments which are involved in the query (based on the transitive closure) are determined by interpreting some sort of "fragment connection graph". Second, the – probably slightly modified – query is executed at each fragment in parallel. At last the partial results are combined according to the initial query. This method works well for e.g. the connection and shortest path problems. However, because of the quite imprecise connection information at the fragment level, the execution of multi-level expands using this approach would result in too many objects in the intermediate results which would have to be filtered out in the final "merging" step. To avoid this we store the precise information about the connected *objects* of different fragments within the OLL catalog.

At first glance, the OLL catalogs may look like the routing tables of computer networks ([Tan89]). Routing tables are used to route network packets between two nodes via the cheapest or shortest path. In contrast to our OLL catalog initialization, algorithms computing such tables typically use some sort of shortest path approaches.

OLL catalogs may bee seen as an index along path expressions, like a nested or path index (cf. [YM98]). Path indexes support the efficient evaluation of conditions in WHERE-clauses of queries. The benefit of path indexes is based on the fact that the evaluation of conditions in WHERE-clauses of queries does not need to traverse substructures of requested objects. In our case, however, queries such as the multi-level expand are to return an entire substructure by traversing it. In order to speed up this different kind of queries, we need an other type of index. OLL catalog entries do not depend on *values* of object attributes – in contrast to nested or path indexes – but on the *location* of the objects.

In this paper we regard product structures as a special kind of directed acyclic graphs. All algorithms presented here are specialized versions of tree traversals using the depth-first search (cf. [CLR96], [Sed88]).

Partitioning directed acyclic condition graphs seems to be similar to the derived horizontal partitioning of tables in distributed databases ([CP84]). However, efficient query processing in this special context has also received little attention only.

Analytical computations (cf. [MDEF01]) have shown that in most cases (assuming realistic product structures) the algorithms described in section 5 may eliminate up to 95% of the original response times of multi-level expand actions! Figure 4 illustrates the typical results of such computations: The response times of multi-level expand actions using the simple "traditional" approach are compared to those using the OLL approach on the basis of three differently sized product structures in three different network environments (the assumed latency times T_{lat} and data transfer rates dtr of configurations 1, 2 and 3 are $T_{lat} = 150ms$ and $dtr = 256kBit/s$, $T_{lat} = 150ms$ and $dtr = 512kBit/s$, and $T_{lat} = 50ms$ and $dtr = 1MBit/s$ respectively).

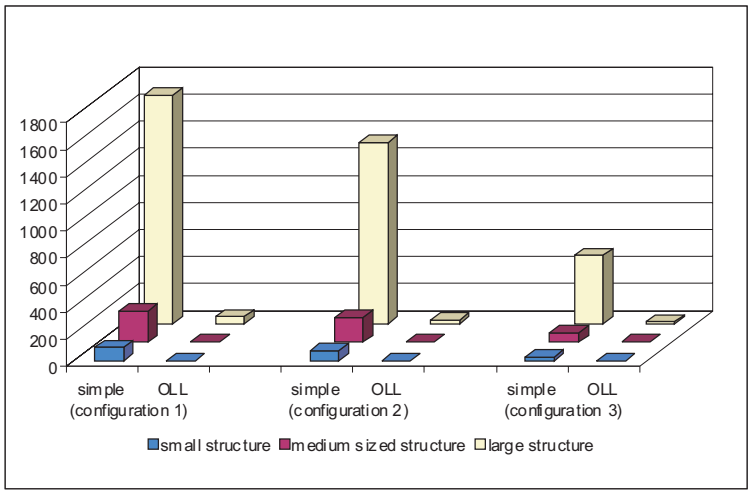


Figure 4: Multi-level expand response times of three different product structures in three different network environments

The OLL approach has been implemented experimentally using IBM DB2 UDB v7.2 as the underlying DBMS. The measurements performed showed response times which came very close to the analytically estimated ones.

7 Summary and Outlook

In order to survive severe competition, collaborative product development spanning several geographically distributed partners is becoming an indispensible must for manufacturing industries more and more. Even in wide area networks, fast access to the partners' data – especially for assembling product structures – is essential for efficient collaboration.

In this paper we introduced OLL catalogs. In short, they store information about transitive dependencies of parts, which are stored in different partitions at possibly different database servers. With these catalogs we are able to minimize the number of communications between involved database servers: Each server is contacted at most once, and even parts that occur more than once within the product structure are queried only once. As no intermediate results are necessary to generate queries that retrieve missing remote substructures, all queries can be executed in parallel, thus further minimizing the response times of assembly operations.

The OLL approach described in this paper shows the direction into which PDM systems have to move in order to meet the performance requirements of worldwide distributed product development.

References

[ANS99] ANSI/ISO/IEC 9075-2:1999 (E). *Database Language SQL – Part 2: Foundation (SQL/Foundation)*, September 1999.

[CCH93] Filippo Cacace, Stefano Ceri, and Maurice A. W. Houtsma. A Survey of Parallel Execution Strategies for Transitive Closure and Logic Programs. *Distributed and Parallel Databases*, 1(4):337–382, 1993.

[CIM97] CIMdata, Inc., CIMdata World Headquarters, Ann Arbor, MI 48108 USA. *Product Data Management: The Definition. An Introduction to Concepts, Benefits, and Terminology*, fourth edition, September 1997.

[CLR96] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT-Press, 1996.

[CP84] S. Ceri and G. Pelagatti. *Distributed Databases. Principles and Systems*. McGraw-Hill, 1984.

[EM99] A. Eisenberg and J. Melton. SQL:1999, formerly known as SQL3. *ACM SIGMOD Record*, 28(1):131–138, March 1999.

[Fit90] M. C. Fitting. *First-order logic and automated theorem proving*. Springer, New York, Heidelberg, 1990.

[Gra93] G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[HAC90] M. Houtsma, P. Apers, and S. Ceri. Distributed Transitive Closure Computations: The Disconnection Set Approach. In *Proceedings of the 16th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Brisbane*, 1990.

[HMS92] T. Härder, B. Mitschang, and H. Schöning. Query Processing for Complex Objects. *Data & Knowledge Engineering*, 7:181–200, 1992.

[IBM01] IBM Corporation. *IBM DB2 Universal Database – SQL Reference – Version 7*, 2001.

[KGM91] T. Keller, G. Graefe, and D. Maier. Efficient Assembly of Complex Objects. In *Proceedings of ACM Sigmod Conference, Denver, Colorado, May 1991*, volume 20, pages 148–157. ACM Press, June 1991.

[Kos00] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Computing Surveys*, September 2000.

[LE78] A. H. Lightstone and H. B. Enderton. *Mathematical logic: An introduction to model theory*. Plenum Pr. New York, 1978.

[MDEF01] E. Müller, P. Dadam, J. Enderle, and M. Feltes. Tuning an SQL-Based PDM System in a Worldwide Client/Server Environment. In *Proceedings of 17th International Conference on Data Engineering, Heidelberg, Germany*, pages 99–108, 2001.

[MGS+94] D. Maier, G. Graefe, L. Shapiro, S. Daniels, T. Keller, and B. Vance. Issues in Distributed Object Assembly. In M. T. Özsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management (Proceedings of the 1992 International Workshop on Distributed Object Management, August 1992, Edmonton, Canada)*, pages 165–181. Morgan Kaufmann Publishers, 1994.

[MOn] Matrix One. www.matrix-one.com.

[MP] Metaphase. http://www.plmsol-eds.com/metaphase/index.shtml.

[NCW93] W. Nejdl, S. Ceri, and G. Wiederhold. Evaluating Recursive Queries in Distributed Databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(1):104–121, February 1993.

[OLR95] A. Obank, P. Leaney, and S. Roberts. Data mangement within a manufacturing organization. *Integrated Manufacturing Systems*, 6(3):37–43, 1995.

[Sed88] R. Sedgewick. *Algorithms*. Addison Wesley, 2nd edition, 1988.

[Tan89] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall International Editions, second edition, 1989.

[WT] Windchill Technologies. http://www.ptc.com/products/windchill/index.htm.en.

[YM98] C. T. Yu and W. Meng. *Principles of Database Query Processing for Advanced Applications*. Morgan Kaufmann Publishers, San Francisco, California, 1998.

Towards Federated Search Based on Web Services

Jens Graupmann, Michael Biwer, Patrick Zimmer
University of Saarland, Germany
Department of Computer Science
P.O. Box 151150, D-66041 Saarbrücken
E-mail:graupman@cs.uni-sb.de

Abstract: Some emerging trends in the recent development of the WWW can be observed. These trends are technical, like Web Services, as well as semantic, like the integration of ontologies. We propose an architecture for a new kind of federated search system, which takes these aspects and new developments into account. A major challenge in this context is to cope with portals and the data sources behind the portals, the so-called "Deep Web". One component of the proposed architecture is the service mediator, which generates wrapper classes and additional files to make portals accessible as Web Services. Other components are an onotology server, which provides Web Service based access to different ontologies and an XML Filter server that converts different source formats to XML. This loosely coupled architecture supports federated search on semistructured data and the evaluation of semantic join operations.

1 Introduction

1.1 Motivation

Some emerging trends in the recent development of the WWW can be observed. These trends are technical, like Web Services, as well as semantic, like the integration of ontologies. They are the foundation of the Next Generation Web. We will point out some aspects of these developments and propose as an application example an architecture for a new kind of federated search system, which takes these aspects and new developments into account. One of the differences to other systems is that we do not want to do any kind of schema integration . We don't even know the data format of some of our data sources in advance. We only want to nd data objects (HTML, XML, PDF documents etc.) satisfying our search conditions as well as possible. First we will introduce some of the emerging technologies.

1.1.1 The "Deep Web" (Portals)

A lot of information in the WWW is stored in data sources, mostly relational databases, which are connected to some server application logic, which dynamically generates Web pages. These information repositories are hosted by so-called *Web Portals*. This highly

dynamic content leads to problems with today's prevalent crawler based search engines. The main issues are:

1. Information is not accessible by crawlers, because it is generated as a response to a HTML form submission.
2. Links between pages change frequently. For example the content of a category daily news changes daily and is regularly moved to the archive category.

To avoid these problems we wrap the portal search engines themselves into Web Services and use them in the fashion of a meta search engine.

1.1.2 The "Semantic Web" (Ontologies)

One important aspect needed to improve the accuracy of search engines is the inclusion of ontological information, not only to analyze Web content, but also to interpret and expand user queries. The Next Generation Web should be able to provide this information by itself. It has to provide a kind of semantic interoperability [DMHF00]. An approach to provide means formalizing this is RDF/RDF-Schema [RDF00], a language to describe Web resources and the corresponding schema containing (weak) ontological information. A more expressive language for this purpose is DAML+OIL [DAML02], which seems to become a de-facto standard.

1.1.3 The "Automated Web" (Web Services)

Web pages in general are generated by humans for humans. This is of course not surprising, but if we want to use tools to ef ciently support us, we have to utilize technologies, which allow computer programs to interact ef ciently by using an appropriate data format, as well as an appropriate infrastructure. In many projects such infrastructures and data exchange formats were invented, but only recently these technologies are becoming widely accepted. These technologies include, of course, XML as data exchange format and Web Services as infrastructure. SOAP is the Protocol on top of HTTP, the services themselves are described in the Web Service Description Language WSDL and registered in the Universal Description, Discovery and Integration (UDDI) Registry [UDDI01].

1.2 Related Work

Several recent approaches have dealt with the integration of heterogeneous data sources. One of them is the TSIMMIS project [CHW91] . Its architecture is based on data source wrappers, which are generated semi automatically. The wrappers, called Translators, convert the data sources into a common data format. The Carnot[CHW91] project was one of the rst projects dealing with a federated search architecture integrating ontologies. In this project a general purpose ontology is used. The OBSERVER Project [MKSI96] integrates

multiple ontologies and considers problems, arising in the context of the combination of the different ontologies (mapping, inconsistencies etc.). Every node in the system has a component, called Ontology Server, which is responsible for the translation of the query expressed in terms of a user ontology (the ontology, the user has chosen before formulating the query) into queries understood by the underlying data repositories. This translation is done by utilization of mapping information de ned for every data repository. In [BNL98] the topic of Join Processing in Web Databases is addressed. In this work rst a database consisting of Web pages and links is built. Based on these materialized views a join can be computed on the link structure. This means that Web tuples from one table are connected to the tuples of another table, when pointing to the same URL. In contrast W3QL [KOSH98] considers the WWW itself as a large database and uses an SQL like query language for query processing.

1.3 Contribution

Most recent approaches have not addressed the issue of uncertainty in the quality of the results and data sources. But even those approaches that have considered uncertainty, have not considered join operations on multiple uncertain data sources. They have also posed stringent demands on the integrated data sources to permit a uni ed view on them, mostly realized by schema integration. In this work we introduce a federated search architecture, which aims to seamlessly integrate existing data sources, especially (Deep) Web Portals based on emerging technologies like Web Services and Web Ontologies. Furthermore we will outline how to compute a similarity join operation on these uncertain results and point out some aspects that have to be considered. Additional components are introduced for ontological support and the (pre)processing of the different data formats in order to deal more effectively with various other heterogeneous data sources.

2 Architecture

2.1 "Automated Web" meets "Deep Web"

Towards a Federated Service Web

In this section we describe a federated search architecture based on Web Service technologies. Although these techniques are rooted in the B2B area for commercial transactions [CS02], they can be adapted for other purposes, too. Figure 1 depicts an application scenario. First a client submits a query to a special kind of meta search engine. This engine should be able to condense the query to a term pair consisting of one concept or class and a set of properties for one instance of this class. In the example shown in gure 1 the submitted query deals with *book* s about the topic *repairing Audi cars* . This pair is submitted to the service search engine (1). The class term is used by the search engine

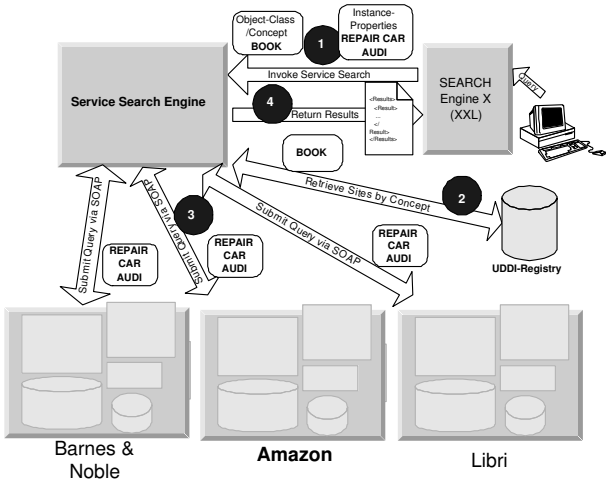


Figure 1: Web Service search

to determine relevant service providers (2). Once relevant services are found, the instance properties are submitted to these service providers (3) and results are received, combined and submitted back (4) to the rst search engine.

Leveraging Web Services Infrastructure

Most existing Web Portals do not support Web Services, but they provide HTML form based access to an internal search engine. One way to integrate such servers is the generation of wrappers, which are hosted by a dedicated server application. Such an application, called *Service Mediator*, is responsible for the generation and invocation of wrapped services. If a client application gets a reference to a Web Service, hosted by the *Service Mediator,* it retrieves the corresponding WSDL description and generates an appropriate SOAP Message to invoke the service. The translation layer of the service mediator interprets the SOAP message, activates the corresponding wrapper classes and invokes the necessary methods to communicate via HTTP/HTML with the wrapped portal site to invoke the internal search engine of

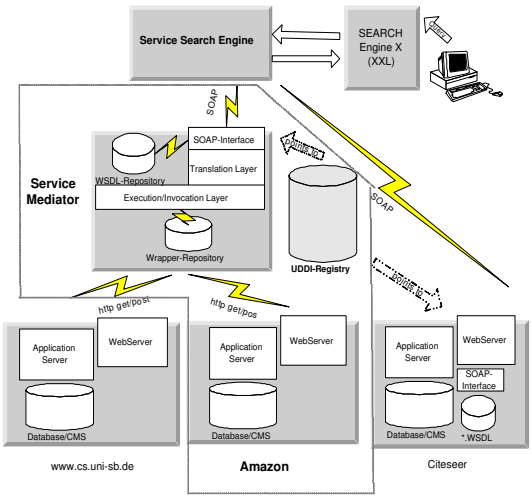


Figure 2: Service Mediator at runtime

the portal.

Figure 2 depicts a scenario with three portals. The two portals on the left do not support Web Services directly. Access via Web Services is provided by the Service Mediator. The portal on the right (Citeseer) is fully service enabled [1]. Therefore the Service Mediator is not needed to access its data. The generation of a wrapper for a speci c Web site is initiated by a special SOAP message (Figure 3). This message contains the target portal's URL and additional information needed later for registration. In this generating mode, the system tries to parse the target Web site(1) and to detect form elds. Labels and internal names of the form elds are used as parameter names for the WSDL description. In many cases it might be possible to automatically generate a wrapper class for this site, but in complicated cases, especially if the target site uses script technologies manual interaction is probably indispensable. Furthermore the transformation of the result pages to XML can hardly be done automatically. For this purpose some frameworks have been developed, e.g. the W4F Toolkit[SA99], using a proprietary extraction and transformation language and the Andes Toolkit[Myl01] using XML Technologies, especially XSLT. After the generation of the wrapper classes and the corresponding WSDL description (2), the wrapped service is registered with a (UDDI-) registry (3) (This has not necessarily to be a UDDI Registry). The keywords and categories for the registration are submitted by the application, which initiated the wrapper generation. That means that this application has to do a preprocessing of the portal to extract meaningful keywords for categorization.

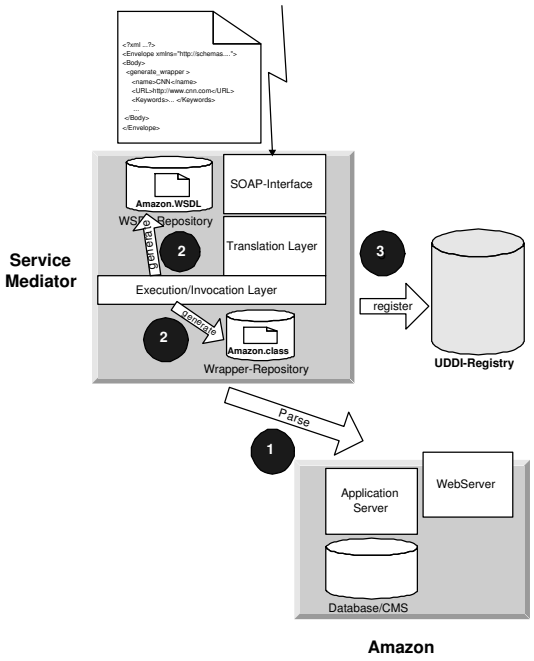


Figure 3: Service Mediator at generation time

2.2 "Semantic Web" meets "Deep Web"

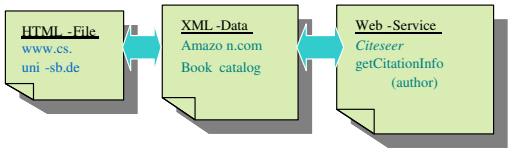
We want to enhance this architecture by the integration of Semantic Web technologies that are already in use in different application areas [FBDK02]. This means precisely that we integrate metadata especially ontologies into query formulation as well as query processing. The aspect of ontology based query formulation is addressed in the next section. From the architecture point of view we treat ontology sources like any other data source,

[1] Currently Citeseer does not offer such an interface. This is a fictitous example.

that we integrate with a Web Service interface. Currently we are implementing an ontology server which provides uni ed access to WordNet [Fel98], OpenCyc [OC02] and DAML+OIL [DAML02]. The returned results also contain scores to express the similarity of the returned descriptions to the original concept. These scores are needed for the computation of the overall similarity score of the returned values. The ontologies are connected to the server via a speci c adapter which has to be implemented for every native ontology API.

2.3 Integration of Heterogeneous Data Sources

Due to the fact, that many information sources do not provide any metadata, we have to analyze the data and infer metadata to apply our ontologies to these data sources. One example for the generation of such metadata is the matching of concepts and attributes of an ontology to text fragments in a semistructured text document, e.g. the identi cation of book descriptions in an HTML page. In general automatic analysis is error prone. So we can not be sure about the correctness of our generated metadata. Even if the information source provides metadata, this often does not match the metadata our query was based on, e.g. one concept in the ontology, we used to pose our query does not match any concept of the ontology of the data source, but there may be many similar concepts. In consequence we have to deal with uncertainty and similarity, where results are ranked according to their quality . To compute these values and subsequent to determine the best matches for the whole query, we have to build virtual relations which are views on our data objects, containing only the attributes stated in the query together with the corresponding similarity values. On the other hand if a data source is already metadata enabled, like the RDF+OIL enabled portal presented in [VDGA00], which also includes an interface for the RDF query language (RQL), the system can be integrated more easily. In the example shown below we have to compute a join between a HTML page and XML data.



Suppose the join attribute is a person name extracted from the Web page, which we want to connect to an author of a book contained in our XML book data. Furthermore we want to obtain citations of this author using a Web Service method provided by *Citeseer*. Other conceivable applications arise in corporate networks, where diverse information is distributed over relatively unstructured information sources e.g. folders containing spreadsheets, Word documents or Personal Information Manangement (PIM) applications. In this case the wrapper has to implement a whole subsystem to provide ef cient access to these documents. In these cases on-demand retrieval is also crucial, because the freshness of the gathered information can be economically and legally important.

In conclusion we have to deal with the following dimensions of heterogeneity.

- **Structure**: We have to deal with unstructured data (plain text), semi structured data (like XML) and structured data (Database relations)
- **Metadata**: Data sources can contain no metadata, simple metadata (*Dublin core* metadata in HTML pages) and complex meta data (ontologies, RDF/S).
- **Ranking**: Some data sources return a list of unordered results. Other sources rank the results relatively or even with an absolute score representing their quality in respect to the query.
- **Query capabilities**: A data source does not have to provide any query capability at all (collection of text documents on a le system), but may provide a keyword based search or even an interface for a complex query language (SQL or XQuery).

To overcome the problem of structural similarity at least partly we are implementing a framework to convert different data formats to XML as an intermediate format. The XML Filter module of this framework not only converts HTML to XML, but also performs some preprocessing to support further steps. One of the aims of this HTML preprocessing is to make the hidden semantics in HTML documents more explicit, e.g. to convert the content of headings to XML Tags or extract the row and column labels of a HTML table. In Figure 4 all relevant components are shown.

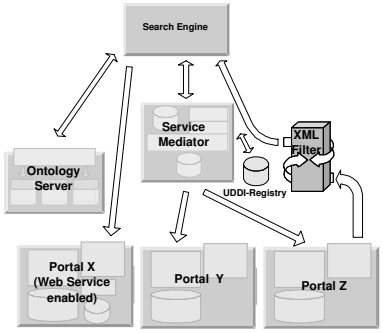


Figure 4: Architecture Overview

3 Prototype Implementation

We have implemented an initial prototype system and are in the process of extending its scope and capabilities. Currently the system automatically analyzes Web pages with HTML forms and generates the corresponding WSDL descriptions and wrapper classes for all forms on this page. The system is implemented in Java and can be invoked using a Web front-end via a Java servlet. The UDDI Registry we use is developed by HP [HP02]. For generation of the WSDL Description we use the UDDI4J package and the WSDL4J Package from IBM [IBM02]. For test purposes we have implemented another Servlet, which transform the WSDL form Description back into a HTML form via XSLT. The HTML frontend of the system allows as well the initiation of the generation, the registration at the registry with keywords, and the deletion of the Web Service as the search and invocation. Figure 5 shows a source HTML form (a), a fragment of the generated WSDL description (b) and the form rebuilt via XSLT (c). To get the rebuilt form, rst the Web Service has to be retrieved from the UDDI Registry via the corresponding call. Next the WSDL description is transformed back by an XSLT script into a HTML form, similar to the original form. When this form is submitted the form data is sent back to the servlet, which parses the submitted data, builds the correct SOAP function call according

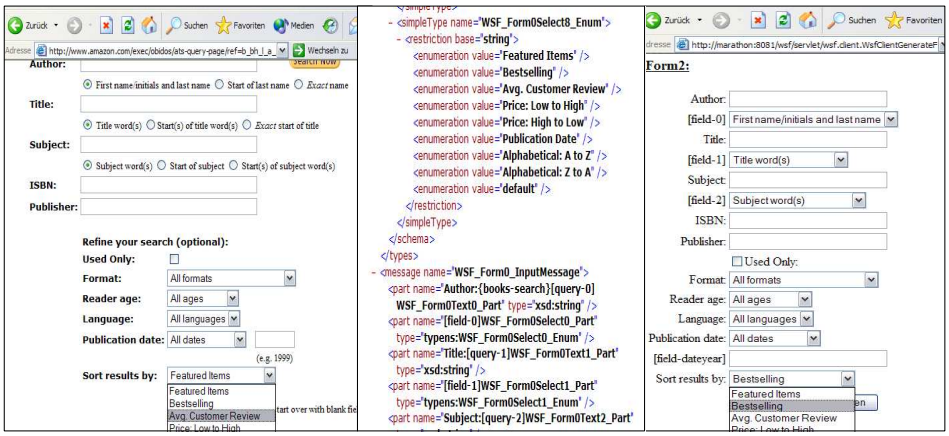


Figure 5: (a) Source Html form, (b) Generated WSDL file, (c) Rebuilt HTML form

to the WSDL description and invokes the corresponding Web Service. The wrapper then tries to send the function arguments to the HTML form in the correct format and returns the result URL to the Web Service caller. For forms using the HTTP POST Method the result is cached in a local directory and this URL is returned to the caller. The test client now simply displays the result URL. This is a very comfortable way to test the generated WSDL descriptions. Of course the search engine invokes the Web Service directly. The parser for the Web pages uses some simple heuristics to extract identi ers for form elds. Of course these can not cover every kind of HTML coding, but work fairly good on most forms. Nevertheless we are considering the use of more advanced machine learning techniques for this purpose (see, e.g., [BDS01]). This wrapper framework allows us to add new data sources almost automatically.

4 Towards Federated Search Based on Web Services and Ontologies

As a consequence of our intention to integrate many information sources with different query capabilities and data formats, we have to consider this different quality of data. A data source which provides XML les and corresponding XML schemas or RDF descriptions is easier to interpret than uncommented HTML pages, which have to be parsed and heuristically analyzed. So we have two main dimensions of uncertainty. The rst is the con dence of the system to have recognized attributes in the data object correctly. For example if we want to extract the mileage of cars from car advertisements, one value for each returned attribute value pair represents the con dence of the system, that this text fragment really corresponds to a the representation of mileage. [EJX01] proposed four different facets of metadata for the computation of these values.

- Terminological relationships (synonyms, hyponyms etc.)

- Data value characteristics (the mileage of car is a number between one and one million)
- Target speci c regular expression matches (if we're looking for the mileage of car in an advertisement we expect the term 'mile[s]' or 'km')
- The structure and sequence of elements.

We expand the terminological metadata facet to ontological metadata because we are interested in the number of recognized attributes of the assumed concept in the data object. The other dimension of uncertainty for this attribute value pair (recognized attribute plus value of this attribute) represents the domain dependent similarity to the value of the lter condition of the query. Based on these similarity values we want to compute a similarity join operation that combines uncertain information of different data sources. Queries are posed based on one or more ontologies. The query itself is formulated in an SQL-style query language, where relations are substituted by concepts of an ontology and schema attributes correspond to concept attributes.

5 Conclusion

We proposed an architecture for a federated search system, which integrates existing information sources, especially Web Portals but also other heterogeneous data sources utilizing state-of-the-art technologies. Most other approaches pose very strict (often only implicitly mentioned) requirements on the data sources, whereas our approach can handle very different data sources. From the outset, we considered different kinds of uncertainty in our architecture. This also allows us to compute a similarity join operation, which connects information from different heterogeneous data sources in a meaningful way [Fag96]. Although our approach integrates onotologies, we can include data sources without ontological support directly, like Web Services in the style of simple Web accessible methods. This exibility mainly roots in the very loose coupling of the components that also allows dynamic recon guration (e.g. addition of new data sources) of the system at runtime.

References

[BNL98] S. S. Bhowmick, W. K. Ng, E.-P. Lim: Join Processing in Web Databases. 9th International Conference on Database and Expert Systems Applications (DEXA'98), 1998

[CGHI94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom: The TSIMMIS Project: Integration of Heterogeneous Information Sources. 16th Meeting of the Information Processing Society of Japan (IPSJ'94), 1994

[CHW91] C. Collet, M. N. Huhns, W.-M. Shen: Resource Integration Using a Large Knowledge Base in Carnot. IEEE Computer 24(12): 55-62, 1991

[CS02] F. Casati, M.-C. Shan: Models and Languages for Describing and Discovering E-Services, SIGMOD Conference 2001 (Tutorial), 2001

[DAML02] The Darpa Agent Markup Language, http://www.daml.org

[DMHF00] S. Decker, S. Melnik, F. v. Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, I. Horrocks: The Semantic Web: The Roles of XML and RDF. IEEE Internet Computing 4(5): 63-74, 2000

[EJX01] D. W. Embley, D. Jackman, L. Xu: Multifaceted Exploitation of Metadata for Attribute Match Discovery in Information Integration. Workshop on Information Integration on the Web (WIIW 2001), 2001

[Fag96] R. Fagin: Combining Fuzzy Information from Multiple Systems. ACM Symposium on Principles of Database Systems (PODS'96), 1996

[FBDK02] D. Fensel, C. Bussler, Y. Ding, V. Kartseva, M. Klein, M. Korotkiy, B. Omelayenko, and R. Siebes: Semantic Web Application Areas. 7th International Workshop on Applications of Natural Language to Information Systems (NLDB 2002), 2002

[HP02] HP total e-server; HP Web Services developer edition, http://www.bluestone.com/

[IBM02] UDDI4J, http://oss.software.ibm.com/developerworks/projects/uddi4j

[KS96] V. Kashyap, A. P. Sheth: Semantic and Schematic Similarities Between Database Objects: A Context-Based Approach. VLDB Journal 5(4): 276-304, 1996

[KOSH98] D. Konopnicki, O. Shmueli: Information Gathering in the World-Wide Web: The W3QL Query Language and the W3QS System. TODS 23(4): 369-410, 1998

[MKSI96] E. Mena, V. Kashyap, A. P. Sheth, A. Illarramendi: OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. International Conference on Cooperative Information Systems (CoopIS 96), 1996

[Myl01] J. Myllymaki: Effective Web data extraction with standard XML technologies. International World Wide Web Conference (WWW 2001), 2001

[OC02] OpenCyc, The Open Source version of Cyc technology, http://www.opencyc.org

[RDF00] Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, http://www.w3.org/RDF/

[SA99] A. Sahuguet, F. Azavant: Looking at the Web through XML Glasses. International Conference on Cooperative Information Systems (CoopIS 1999), 1999

[UDDI01] Universal Description, Discovery and Integration (UDDI), Specification 2.0. http://www.uddi.org

[VDGA00] V. Christophides, D. Plexousakis, G. Karvounarakis & S. Alexaki, Declarative Languages for Querying Portal Catalogs, 1st "DELOS"Workshop on "Information Seeking, Searching and Querying in Digital Libraries", 2000.

[BDS01] V. Borkar, K. Deshmukh,S. Sarawagi: Automatic Segmentation of Text into Structured Records. SIGMOD Conference (SIGMOD 2001), 2001

[Fel98] C. Fellbaum (Editor): WordNet: An Electronic Lexical Database, MIT Press, 1998

Von digitalen Bibliotheken zu Online-Shops mit digitalen Produkten

Dietrich Boles

Universität Oldenburg
Fachbereich Informatik
Escherweg 2
D-26121 Oldenburg
boles@informatik.uni-oldenburg.de

Abstract: Digitale Bibliotheken lassen sich als organisierte Sammlungen digitaler (multimedialer) Dokumente charakterisieren. Sie ermöglichen Anbietern die Verwaltung und Konsumenten den Zugriff auf die Dokumente über das WWW. Online-Shops bilden traditionelle Verkaufswelten und -vorgänge in die Virtualität ab. Sie ermöglichen Anbietern den Verkauf und Konsumenten den Kauf von (materiellen) Produkten über das WWW. So genannte *dShops*, die in diesem Artikel eingeführt werden, kombinieren die Eigenschaften und Funktionalitäten digitaler Bibliotheken und Online-Shops. Aus Sicht digitaler Bibliotheken handelt es sich bei dShops um kostenpflichtige digitale Bibliotheken, aus Sicht des eCommerce um Online-Shops mit digitalen Produkten.

1 Einleitung

Als *Digital Commerce* (*dCommerce*) wird der elektronische Handel mit digitalen Produkten, wie WWW-Inhalte, digitale Audios und Videos, multimediale Dokumente oder Software, über das Internet bezeichnet [Lux00]. Eine charakteristische Eigenschaft, die den dCommerce vom elektronischen Handel mit materiellen Produkten (*Electronic Commerce*, *eCommerce*) unterscheidet, ist die Möglichkeit einer vollkommen digitalen Geschäftsabwicklung inklusive der Distribution. Im Vergleich zum eCommerce ist der dCommerce Ende des Jahres 2002 jedoch noch wenig verbreitet. Insbesondere Web-Inhalte werden heutzutage weitgehend kostenlos zur Verfügung gestellt. Langfristig gesehen und durch aktuelle Prognosen bestätigt ist es jedoch sehr wahrscheinlich, dass der Abruf qualitativ hochwertiger Informationen und die Inanspruchnahme von Dienstleistungen im WWW in steigendem Maße kostenpflichtig werden. [1]

Vor diesem Hintergrund wurden in einem Forschungsprojekt an der Universität Oldenburg ausgehend von einer detaillierten Analyse der konkreten Eigenschaften und Anforderungen des dCommerce Konzepte und Softwaresysteme zum „Experimentieren“ mit dem

[1] vgl. bspw. http://www.heise.de/newsticker/data/hob-17.01.02-000/

dCommerce entwickelt. Zentraler Ansatz war dabei der Ausbau digitaler Bibliotheken, in denen digitale Produkte verwaltet werden, zu so genannten *dShops*, die den Verkauf der digitalen Produkte über das WWW ermöglichen. Beim Ausbau wurden insbesondere Konzepte und Technologien herkömmlicher Online-Shops für materielle Güter berücksichtigt.

Hauptergebnis der Forschungsaktivitäten ist ein Referenzmodell für so genannte *dShop/-dMall-Systeme*. dShop/dMall-Systeme sind spezielle Softwaresysteme, die durch den Einsatz graphisch-interaktiver Hilfsmittel und die Integration erprobter, bewährter und Erfolg versprechender Geschäftspraktiken den Aufbau, die Verwaltung und den Einsatz von dShops auch für Anbieter ohne tief gehende technologische und kaufmännische Fachkenntnisse unterstützen. Das Referenzmodell beschreibt den logischen Aufbau und die generelle Funktionalität solcher dShop-Systeme. Von konkreten bzw. implementierungsspezifischen Details wird abstrahiert. Es zerlegt das Gesamtsystem in einzelne Teilkomponenten und identifiziert deren Schnittstellen. Letztendlich dient es als abstrakte Entwurfsvorlage für die Implementierung konkreter Systeme. Das Referenzmodell basiert auf objektorientierten und komponentenbasierten Modellierungskonzepten sowie praktischen Erfahrungen, die bei der Entwicklung und Erprobung zweier konkreter dShop-Systeme, dem eVerlage-System [BHO00] und dem DDS-System [BH00], gesammelt wurden. Als Notation für die Beschreibung des Referenzmodells wird die Unified Modeling Language (UML) verwendet.

Das Referenzmodell für dShop/dMall-Systeme kombiniert die Funktionalität und Struktur digitaler Bibliotheken und Online-Shops für materielle Produkte, berücksichtigt dabei aber die besonderen Eigenschaften digitaler Produkte und die sich daraus ergebenden besonderen Anforderungen an den Handel mit digitalen Produkten. Insbesondere unterstützt das Referenzmodell speziell für den dCommerce interessante Geschäftspraktiken, wie eine differenzierte Produkt- und Preispolitik, den Handel mit Nutzungsrechten an digitalen Produkten anstelle des Handels mit den Produkten selbst sowie vielfältige Erlösformen über Subskription und Einzeltransaktionen bis hin zu Gruppenlizenzen. Eine besondere Bedeutung kommt zusätzlichen Sicherheitskonzepten beim dCommerce zu, vor allem dem Schutz von Urheberrechten an digitalisiertem geistigen Eigentum.

In diesem Artikel werden in Abschnitt 2 und 3 zunächst digitale Bibliotheken und Online-Shops charakterisiert. Abschnitt 4 geht auf wesentliche Aspekte des dCommerce ein. In Abschnitt 5 wird ausgehend von den Ausführungen in den vorangehenden Abschnitten das Referenzmodell für dShop/dMall-Systeme skizziert. Aus Platzgründen kann jedoch nur ein kleiner Ausschnitt des gesamten Referenzmodells in diesem Artikel vorgestellt werden. Für eine vollständige Beschreibung des Referenzmodells sei auf [Bol02] verwiesen. Der Artikel schließt in Abschnitt 6 mit einer Bewertung und einem Ausblick.

2 Digitale Bibliotheken

Digitale Bibliotheken lassen sich als „organisierte Sammlungen digitaler Dokumente“ charakterisieren [Arm99, EF00]. Sie bilden ein Distributionsmedium zwischen Anbietern von in digitaler Form vorliegenden Informationen und potentiellen Konsumenten der Informa-

tionen. Sie erweitern traditionelle Bibliotheken zum einen durch die Möglichkeit der Speicherung und Verwaltung neuer Informationstypen, wie multimediale Dokumente, und zum anderen durch die Automatisierung, Redefinition und Erweiterung konventioneller Bibliotheksdienste.

Im engeren Sinne der Definition handelt es sich bei den digitalen Dokumenten um textuell-graphische Dokumente, die in digitalen Bibliotheken im Unterschied zu traditionellen Bibliotheken nicht mehr in papierner sondern in digitaler Form vorliegen. Im weiteren Sinne kann es sich bei den Dokumenten jedoch auch bspw. um digitale Videos, Audios, Multimedia-Applikationen, Software oder Geodaten handeln. Digitale Bibliotheken garantieren eine langfristige Speicherung der digitalen Dokumente. Die Speicherung basiert dabei auf einem Dokumentenmodell, das sowohl die interne Struktur der Dokumente als auch Beziehungen zwischen Dokumenten (externe Struktur) definiert. Zusätzlich zu den eigentlichen Dokumenten selbst werden in so genannten Metadaten weitere Beschreibungsmerkmale der Dokumente vorgehalten.

3 eShops

Als *Electronic Commerce* wird der Handel von Gütern über Kommunikationsnetze, insbesondere das Internet bezeichnet [Mer02]. Bei den Gütern handelt es sich heutzutage vorwiegend um materielle Güter, wie Bücher, CDs oder andere Gebrauchsgegenstände. Der Verkauf dieser Güter an Endkunden ist über spezielle WWW-Sites, so genannte *Online-Shops* oder *eShops* möglich. Häufig werden eShops zu virtuellen Einkaufszentren, so genannte *eMalls*, zusammengeschlossen. Die Funktionalität von eShops basiert eigentlich immer auf demselben Muster, das traditionellen Einkaufsvorgängen nachgebildet ist: Kunden können entweder über einen Katalog im Produktbestand navigieren oder gezielt nach Produkten suchen. Aus den Trefferlisten lassen sich Produkte auswählen und ausführlichere Beschreibungen (Metadaten), häufig mit Fotos oder Filmen versehen, abrufen. Produkte, die der Kunde erwerben möchte, können in einen virtuellen Warenkorb abgelegt werden. Hat der Kunde seine Auswahl abgeschlossen, begibt er sich zur „Kasse“. Hier sind die Zahlungsformalitäten zu erledigen. Anschließend werden die erworbenen Produkte im Allgemeinen per Post ausgeliefert.

eShop/eMall-Systeme sind Softwaresysteme, die den Aufbau, die Verwaltung und den Einsatz von eShops bzw. eMalls unterstützen. Sie stellen dazu graphisch-interaktive Hilfsmittel zur Verfügung, um es auch Anbietern ohne Programmierkenntnisse zu ermöglichen, Produkte über das Internet zu verkaufen. Sie bilden wesentliche Elemente realer Einkaufsläden in die virtuelle Welt ab und berücksichtigen dabei erprobte, bewährte und Erfolg versprechende Geschäftspraktiken, so dass es auch Anbietern ohne bisherige kaufmännische Erfahrungen gelingen kann, zumindest im kleinen Rahmen ein virtuelles Geschäft erfolgreich zu betreiben.

4 Digital Commerce

Der elektronische Handel mit digitalen Produkten wird als *Digital Commerce* – kurz *d-Commerce* – bezeichnet [Lux00]. Zum Teil wird auch synonym der Begriff *Information Commerce* verwendet. Der dCommerce zeichnet sich dadurch aus, dass die Handelsobjekte digitale Datenmengen sind und die gesamte Geschäftsabwicklung inklusive der Distribution vollkommen digital ohne Medienbruch vorgenommen wird. Zur Durchführung sämtlicher Aktivitäten (Produktion, Vermarktung, Verkauf, Vertrieb, Nutzung) sind lediglich geeignete Hard- und Software sowie insbesondere Netzwerke notwendig. Es ist zu keiner Zeit ein anderes materielles Gut involviert. Auf eine Speicherung der digitalen Daten auf physische Trägermedien (CD-ROM, Diskette) zum Transport wird verzichtet. dCommerce ist also eCommerce mit digitalen Produkten und vollkommen digitalen Geschäftsprozessen. [WSC97] charakterisiert ihn daher auch als „eCommerce in reinster Form“.

Digitale Produkte weisen eine Reihe von Eigenschaften auf, die sie von materiellen Produkten unterscheiden bzw. abgrenzen (vgl. [BR98, Var98, Krc99, Lux00, JP01]). Diese besonderen Eigenschaften haben Auswirkungen auf den (elektronischen) Handel mit digitalen Produkten. Es lassen sich besondere Anforderungen und signifikante Unterschiede zwischen dem dCommerce und dem eCommerce identifizieren:

- **Vollkommen digitale Geschäftsabwicklung**. Aufgrund der Eigenschaft der digitalen Repräsentation digitaler Produkte kann der elektronische Handel mit digitalen Produkten auf vollständig digitalem Wege vollzogen werden, von der Beschaffung des Produktes durch den Händler beim Produzenten, über die Lagerung des Produktes beim Händler bis hin zur Auslieferung des Produktes an die Konsumenten.
- **Einfacher Direktvertrieb**. Mit Hilfe geeigneter Software ist ein Direktvertrieb digitaler Produkte durch den Hersteller wesentlich einfacher zu realisieren, als dies bei materiellen Produkten der Fall ist. Große Warenlager fallen genauso weg, wie Lieferantenverwaltung, Transportlogistik und Retourenhandling.
- **Flexible Angebotspolitik**. Die besonderen Eigenschaften digitaler Produkte erlauben eine sehr viel differenzierte Form des Marketing, insbesondere der Produkt- und Preisgestaltung (siehe auch [Bra00]).
- **Handel mit Nutzungsrechten**. Handelsobjekte im dCommerce müssen nicht unbedingt die digitalen Produkte selbst sein. Vielmehr ist es möglich, mit Nutzungsrechten an digitalen Produkten zu handeln. Ein Kunde erwirbt hierbei bestimmte Lizenzen, die u.a. die befugten Nutzer, die erlaubten Nutzungsformen sowie Nutzungsdauer bzw. -anzahl festlegen. Eingesetzt werden diesbezüglich so genannte *Digital-Rights-Managements-Systeme* [Ste97, Ian01].
- **Vielfältige Erlösformen**. Die Erlösform legt die Finanzierung einer Geschäftstätigkeit fest. Sie beinhaltet, von wem, wann und wofür Einnahmen erwartet werden. Bei den direkten Erlösformen stammen die Einnahmen von den Kunden selbst. Wenn ein Kunde ein Produkt erwerben bzw. in einer bestimmten Form nutzen will, muss er dafür bezahlen. Nutzungsabhängige direkte Erlösformen sind transaktionsbasiert, d.h. jede einzelne Nutzung wird auf bestimmte Art abgerechnet (z.B. Pay-per-View). Hingegen kennzeichnet die nutzungsunabhängigen direkten Erlösformen,

dass der Kunde eine pauschale Gebühr entrichten muss (Subskription). Ein großes Problem des dCommerce ist, dass WWW-Nutzer bisher kaum bereit sind, für „Content“ zu bezahlen. Von daher müssen Anbieter alternative indirekte Einnahmequellen finden. Diesbezüglich bietet sich bspw. die Integration von Werbung nicht nur in die virtuellen Verkaufsräume, sondern unter Umständen sogar in die digitalen Produkte selbst an. Andere indirekte Erlösformen sind die Unterstützung der Bildung kommerzieller Gruppen mit besonderen Gruppenkonditionen bzw. Gruppenlizenzen (Campuslizenzen, Gleitlizenzen) oder die Einführung von Gruppenkonten. Gruppenadministratoren können einen bestimmten Geldbetrag auf ein solches Gruppenkonto einzahlen und den Gruppenmitgliedern erlauben, bis zu einem pro Person festgelegten Maximalbetrag das Geld selbst für den Erwerb von digitalen Produkten bzw. von Lizenzen für digitale Produkte auszugeben.

- **Komplexere Einkaufsmodelle**. Während im eCommerce die Phasen eines Einkaufsablaufs im Allgemeinen sequentiell durchlaufen werden, kann die Geschäftsabwicklung im dCommerce je nach Angebotsform sehr viel komplexere Formen annehmen (vgl. auch [KGMP96, AW01]).
- **Zusätzliche Sicherheitsrisiken**. Die einfache Reproduktion, Distribution und Manipulation digitaler Produkte führen aus Anbietersicht zu zusätzlichen Sicherheitsproblemen. Verstöße gegen rechtliche Bestimmungen, gegen das Urheberrecht oder vertraglich festgelegte Nutzungsvereinbarungen durch die Kunden sind kaum kontrollierbar oder bedingen aufwändige Schutzmaßnahmen.

Die besonderen Eigenschaften digitaler Produkte und die sich daraus ergebenden besonderen Anforderungen an den Handel mit digitalen Produkten führen dazu, dass eShop/eMall-Systeme nicht ohne Weiteres auch für den Verkauf digitaler Produkte über das Internet eingesetzt werden können. Hierfür werden spezialisierte Handelsinformationssysteme benötigt, die auch als *dShop/dMall-Systeme* bezeichnet werden.

5 Referenzmodell für dShops/dMall-Systeme

Aus Sicht digitaler Bibliotheken ist ein dShop/dMall-System ein Softwaresystem zur Verwaltung und zum Betrieb einer kostenpflichtigen digitalen Bibliothek. Aus Sicht des eCommerce ist ein dShop/dMall-System ein spezielles Shop-System für die Vermarktung und den Vertrieb digitaler Produkte. Ein dShop/dMall-System vereint damit Funktionen von digitalen Bibliotheken und eShop/dMall-Systemen. Diese Erkenntnis fließt in die Entwicklung eines Referenzmodells für dShop/dMall-Systeme ein, das in diesem Abschnitt ausschnittsweise vorgestellt wird (siehe auch [Bol02]). Es ergibt sich als Kombination zuvor entwickelter Referenzmodelle für digitale Bibliotheken und für eShop/eMall-Systeme, berücksichtigt dabei aber die spezifischen Eigenschaften des dCommerce (siehe Abschnitt 4). Die entwickelten Referenzmodelle sind dabei sehr viel konkreter und implementierungsnäher als existierende Referenzmodelle für digitale Bibliotheken (vgl. [EF00]), eShops (vgl. [Mer02]) und dShops (vgl. [Lux00]). Im Fokus des Referenzmodells für dShop/dMall-Systeme steht der elektronische Handel mit digitalen Online-Produkten. Als

digitale Online-Produkte werden digitale Produkte bezeichnet, die zentral auf dem Server eines Anbieters gespeichert sind. Dieser gewährleistet den Käufern einen Zugriff auf erworbene Produkte, verhindert jedoch eine dauerhafte Abspeicherung auf den Rechnern der Konsumenten.

Ebenso wie die beiden anderen Referenzmodelle ist das Referenzmodell für dShop/dMall-Systeme komponentenartig aufgebaut. Einige Komponenten konnten direkt übernommen werden, andere mussten an die Spezifika des dCommerce angepasst werden. Abbildung 1 skizziert das Klassendiagramm des Referenzmodells für dShop/dMall-Systeme. Es enthält die Klassen und die Beziehungen zwischen den Klassen. Die graue Hinterlegung deutet den Zuständigkeitsbereich der jeweiligen Komponenten an.

Jedes (kostenpflichtige) Dokument gehört zu einem dShop, für den jeweils ein Anbieter verantwortlich ist. Anbieter können mehrere dShops im Rahmen einer dMall betreiben. Jedem dShop ist ein Dokumentenbestand mit mehreren Dokumenten zugeordnet. Die Klasse `Anbieter` ist als Unterklasse der Klasse `Einzelkunde` modelliert. Hierdurch wird erreicht, dass Anbieter als spezielle Kunden mit einer erweiterten Funktionalität (nämlich der Verwaltungsfunktionalität) angesehen werden können.

Wie in digitalen Bibliotheken auch besitzen Dokumente eine interne und externe Struktur. (Lese-)Proben von Dokumenten lassen sich als spezielle externe Beziehungen repräsentieren. Aus dem eCommerce übernommen wurde die Möglichkeit, Dokumentvarianten definieren zu können, um damit dem Aspekt der Produktdifferenzierung im dCommerce Rechnung tragen zu können.

Jedem Dokument können Metadatensätze zugeordnet werden. Die Klasse `Metadatensatz` ist abstrakt und muss bei einer Umsetzung des Referenzmodells entsprechend der jeweiligen Anforderungen konkretisiert werden. Dokumente lassen sich Kategorien zuordnen, die zu bestimmten Klassifikationsschemata gehören. Dies kann einem Produktkatalog gleichgesetzt werden.

Es wird eine abstrakte Klasse `Kunde` eingeführt, von der die konkreten Klassen `Einzelkunde` und `Gruppe` abgeleitet sind. Das bedeutet, als Kunden werden sowohl (normale) Einzelkunden als auch Gruppen(-kunden) angesehen. Vorteil dieser Modellierung ist, dass sehr einfach Gruppenarbeitskonzepte realisiert werden können, indem Beziehungen von anderen Klassen nicht zur Klasse `Einzelkunde`, sondern zur Klasse `Kunde` führen:

- Archive und Profile können nicht nur Einzelkunden gehören, sondern auch gesamten Gruppen (Gruppenarchive, Gruppenprofile).
- (Gruppen-)Anfragen können durch Experten innerhalb einer Gruppe formuliert und gestellt werden. Die Ergebnisse stehen dann allen Gruppenmitgliedern zur Verfügung.
- Auch Gruppen besitzen einen Warenkorb. Darüber können Gruppenlizenzen bestellt werden.
- Auch Gruppen ist ein Konto zugeordnet, nämlich ein Gruppenkonto.

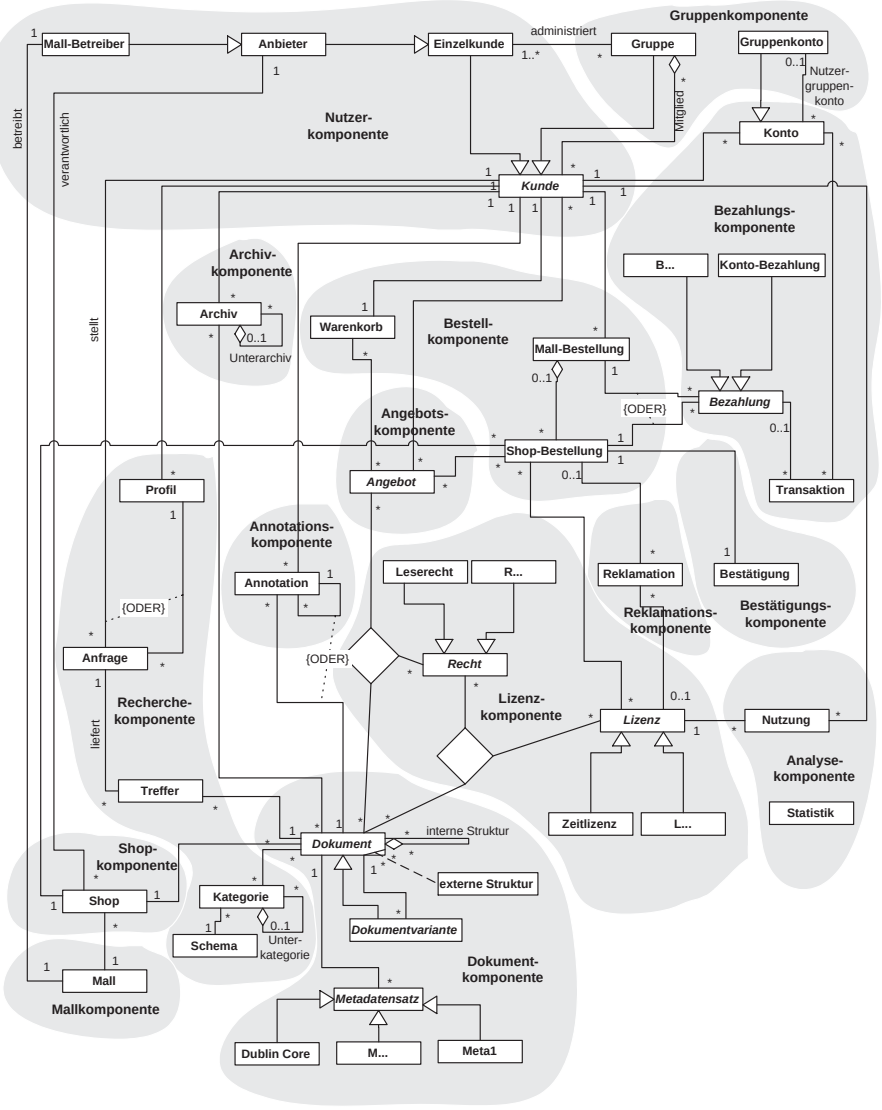


Abbildung 1: Klassendiagramm eines dMall-Systems

Jeder Gruppe sind ein oder mehrere Einzelkunden als Gruppenadministratoren zugeordnet. Durch die Aggregationsbeziehung zwischen den Klassen `Gruppe` und `Kunde` wird sowohl die Gruppenmitgliedschaft als auch die Gruppenstruktur modelliert. Die Aggregation drückt aus, dass Gruppenmitglieder sowohl Einzelkunden als auch wiederum Grup-

pen – Untergruppen – sein können. Jeder Einzelkunde bzw. jede Gruppe kann mehreren Gruppen angehören. Es wird damit eine graphbasierte Gruppenstruktur unterstützt.

Kunden können entweder direkt oder über Profile Suchanfragen stellen, die Mengen von Suchtreffern – Verweise auf Dokumente – produzieren. Kunden besitzen hierarchisch strukturierte Archive, denen Dokumente zugeordnet werden können. Annotationen werden von Kunden erstellt und beziehen sich auf ein Dokument oder eine andere Annotation.

Dokumente (genauer die Angebote) können in einen kundenspezifischen, Mall-übergreifenden virtuellen Warenkorb gelegt werden. Aus diesem heraus wird eine Bestellung initiiert. Eine komplette Bestellung besteht dabei aus mehreren Teilbestellungen, nämlich einer Bestellung pro Anbieter, der von der Bestellung tangiert ist. Neben der Bezahlung gehören eine Auftragsbestätigung und gegebenenfalls Reklamationen zu jeder Teilbestellung.

Für eine Gesamtbestellung oder einzelne Teilbestellungen separat werden Bezahlungsvorgänge initiiert, die je nach Zahlungsverfahren unterschiedlich gestaltet sein können. Alle Zahlungstransaktionen werden gespeichert. Einem Kunden sind unter Umständen mehrere Konten zugeordnet und ein Konto kann Beziehungen zu anderen Konten aufweisen. Hierdurch wird das Konzept der Gruppenkonten umgesetzt. Jedem Kunden können neben seinem normalen Konto so genannte *Nutzergruppenkonten* gehören, die jeweils einem Gruppenkonto zugeordnet sind. Damit kann der Gesamtbetrag eines Gruppenkontos auf die zugehörigen Nutzergruppenkonten aufgeteilt werden, so dass ein Einzelkunde mit Mitteln bezahlen kann, die ihm ein Gruppenadministrator zur Verfügung stellt.

Angeboten werden in dMalls im Unterschied zu eMalls keine Produkte an sich sondern Nutzungsrechte an Dokumenten. Die Klasse `Recht` ist dabei als abstrakte Klasse definiert, die bei einer Umsetzung des Referenzmodells konkretisiert werden muss. Nimmt ein Kunde ein Angebot wahr und leitet eine Bestellung ein, werden letztlich Lizenzen erzeugt, die aus den Rechtespezifikationen der Angebote abgeleitet werden. Lizenzen drücken Nutzungsrechte an Dokumenten aus. Ebenso wie die Klasse `Recht` ist auch die Klasse `Lizenz` abstrakt. Konkretisierungen der Klasse `Lizenz` sind dabei abhängig von Konkretisierungen der Klasse `Recht`. Zugriffe auf Lizenzen werden in der Klasse `Nutzung` festgehalten.

Aufgrund der Flexibilität der Angebotspolitik im dCommerce ist auch die Klasse `Angebot` abstrakt definiert. Im Realisierungsfall muss sie bezüglich der tatsächlichen Angebotspolitik konkretisiert werden. Hierzu enthält sie ein komplexes Attribut `constraints`. Konkrete Angebotsklassen können das Attribut entsprechend nutzen. Constraints können dabei viele Aspekte beinhalten, wie Zeit, Qualität, Menge oder Rabatte.

Die genauen Funktionalitäten der einzelnen Komponenten werden in [Bol02] mit Hilfe von UML-Anwendungsfalldiagrammen beschrieben. UML-Klassendiagramme modellieren insbesondere die anfallenden und zu speichernden Daten und deren Abhängigkeiten voneinander. Über UML-Komponentendiagramme werden Beziehungen zwischen den Funktionen und den Daten hergestellt sowie externe und interne Schnittstellen der Komponenten spezifiziert. Außerdem bereiten die Komponentendiagramme bereits eine mögliche Implementierung auf der Basis der J2EE (Java 2 Enterprise Edition) vor. UML-Sequenzdiagramme dienen als Notation für die Spezifikation unterschiedlicher Einkaufs-

modelle. Diese geben den Durchlauf der einzelnen Phasen (vor allem Bestellung, Bezahlung, Bestätigung und Nutzung) einer Geschäftsabwicklung auf der Basis eines konkreten Geschäftsmodells (Subskription, Pay-per-View, Shareware, ...) an.

6 Bewertung und Ausblick

Zwei dShop-Systeme, die in Anlehnung an das in diesem Artikel skizzierte Referenzmodell für dShop/dMall-Systeme implementiert worden sind, sind das eVerlage-System [BHO00] und das DDS-System [BH00]. Das Referenzmodell kann als konkrete Entwurfsvorlage zur Implementierung weiterer Softwarewerkzeugen für den dCommerce genutzt werden. Wie eine ausführliche Validierung in [Bol02] ergeben hat, erfüllt es nicht nur die besonderen Anforderungen des dCommerce, sondern ist durch seinen komponentenbasierten Aufbau auch sehr flexibel einsetzbar, bspw. in Form einer Kopplung mit existierenden digitalen Bibliotheken, um diese kostenpflichtig zu machen, oder auch in Form einer Erweiterung von Content-Management-Systeme um Vermarktungsfunktionalitäten. Anwendungsfelder sind beispielweise die Bereiche Online-Musik-Distribution, Video-on-Demand, Online-Verlagsprodukte, eLearning-Content und Wissensmanagement.

Durch seinen Komponenten- und Framework-artigen Aufbau erlauben das Referenzmodell bzw. auf der Grundlage des Referenzmodells entwickelte Systeme Anbietern digitaler Produkte ein Experimentieren mit dem dCommerce und eine schrittweisen Lösung der Frage, welche der vielfältigen Geschäftsmodelle und Geschäftspraktiken im konkreten Anwendungsfall für einen Anbieter profitabel und gleichzeitig für die Nachfrager kundenfreundlich und nutzbringend sind. Dies kann von Fall zu Fall unterschiedlich sein, so dass hier Flexibilität gefragt ist und keine fest vorgegebenen Strukturen und Geschäftsmodelle, wie sie bei existierenden eShop-Systemen vielfach vorzufinden sind.

Mit all den genannten Eigenschaften bildet das Referenzmodell eine theoretisch fundierte und praxisrelevante Grundlage für eine Ausweitung des dCommerce und kann mit dazu beitragen, dass sich die Prognosen für einen Aufschwung des dCommerce bewahrheiten werden.

Literaturverzeichnis

[Arm99] William Y. Arms: Digital Libraries. MIT Press, 1999.

[AW01] Karl Aberer und Andreas Wombacher: A language for information commerce processes. In Proceedings Third International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems, San Jose, California, USA, 2001. http://www.opelix.org/public/papers/p2001-06.pdf.

[BH00] Dietrich Boles und Cornelia Haber: DDS: Ein Shop-System für den Information Commerce. In Heinrich Jasper, Josef Küng und Gottfried Vossen, Hrsg., Informationssysteme für E-Commerce (Schriftenreihe Informatik), Tagungsband EMISA-2000 Fachtagung, Volume 4 der Schriftenreihe Informatik, Seiten 229–240, Linz,

November 2000. Universitätsverlag Rudolf Trauner. http://www-is.informatik.uni-oldenburg.de/~dibo/paper/dds-emisa2000.pdf.

[BHO00] Dietrich Boles, Cornelia Haber und Frank Oldenettel: Das eVerlage-System: Verwaltung und Bereitstellung kostenpflichtiger hypermedialer Dokumente im Internet. In HMD - Praxis der Wirtschaftsinformatik, Nummer 214, Seiten 23–34, Heidelberg, August 2000. dpunkt.verlag. http://www-is.informatik.uni-oldenburg.de/~dibo/paper/everlage-hmd2000.pdf.

[Bol02] Dietrich Boles: Integration von Konzepten und Technologien des Electronic Commerce in digitale Bibliotheken. dissertation.de, 2002. Zugl. Dissertation Universität Oldenburg. http://www-is.informatik.uni-oldenburg.de/~dibo/paper/index.html#dissertation.

[BR98] Knut Böhle und Ulrich Riehm: Blütenträume - Über Zahlungssysteminnovationen und Internet-Handel in Deutschland. Wissenschaftlicher Bericht FZKA 6161, Forschungszentrum Karlsruhe, Institut für Technologieabschätzung und Systemanalyse (ITAS), Dezember 1998. http://www.itas.fzk.de/deu/Itaslit/bori98a.pdf.

[Bra00] Roman Brandtweiner: Differenzierung und elektronischer Vertrieb digitaler Informationsgüter. Symposion Publishing, Düsseldorf, 2000.

[EF00] Albert Endres und Dieter W. Fellner: Digitale Bibliotheken: Informatik-Lösungen für globale Wissensmärkte. dpunkt-Verlag, Heidelberg, 2000.

[Ian01] Renato Iannella: Digital Rights Management (DRM) Architectures. D-Lib Magazine, 7(6), Juni 2001. http://www.dlib.org/dlib/june01/iannella/06iannella.html.

[JP01] Mehdi Jazayeri und Ivana Podnar: A Business and Domain Model for Information Commerce. In Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34), 2001. http://computer.org/proceedings/hicss/0981/volume%209/09819079abs.htm.

[KGMP96] Steven P. Ketchpel, Hector Garcia-Molina und Andreas Paepcke: Shopping Models: A Flexible Architecture for Information Commerce. In Proceedings of the Second ACM International Conference on Digital Libraries, 1996. http://dbpubs.stanford.edu:8090/pub/1997-72.

[Krc99] Helmut Krcmar: Informationsmanagement. Springer Verlag, 1999.

[Lux00] Redmer Luxem: Digital Commerce: Electronic Commerce mit digitalen Produkten, Volume 3 of Electronic Commerce. Josef Eul Verlag, Lohmar - Köln, 2000. Zugl. Dissertation Universität Münster (Westf.).

[Mer02] Michael Merz: E-Commerce und E-Business: Marktmodelle, Anwendungen und Technologien. dpunkt-Verlag, Heidelberg, 2002.

[Ste97] Mark Stefik: Shifting the Possible: How Trusted Systems and Digital Property Rights Challenge Us to Rethink Digital Publishing and Agents You Can Understand. Berkeley Technology Law Journal, 12(1), 1997. http://www.law.berkeley.edu/journals/btlj/articles/12_1/Stefik/html/reader.html.

[Var98] Hal R. Varian: Markets for Information Goods. Online-Artikel, 1998. http://www.sims.berkeley.edu/~hal/Papers/japan/japan.pdf.

[WSC97] Andrew B. Whinston, Dahle O. Stahl und Soon-Yong Choi: The Economics of Electronic Commerce. MacMillan Publishing Company, 1997.

Konsistenzquantifizierung in Grid-Datenbanksystemen

Lutz Schlesinger

Universität Erlangen-Nürnberg
Lehrstuhl für Datenbanksysteme
Martensstraße 3
D-91058 Erlangen
schlesinger@informatik.uni-erlangen.de

Wolfgang Lehner

Technische Universität Dresden
Arbeitsgruppe Datenbanken
Dürerstraße 26
D-01069 Dresden
lehner@inf.tu-dresden.de

Kurzfassung: Das Konzept des Grid-Computing gewinnt insbesondere bei verteilten Datenbankanwendungen immer mehr an Bedeutung. Ein Grid-Datenbanksystem besteht dabei aus lokal autonomen Systemen, die für globale Datenbankanfragen zur Ausführung komplexer Analysen zusammengeschaltet werden. Eine Datenbankanfrage wird durch eine Auftragsvergabekomponente an ein Mitglied des Grids weitergeleitet. Um die Anfragegeschwindigkeit zu erhöhen, halten im skizzierten Architekturmodell Knoten die Snapshots von Datenquellen anderer Mitglieder lokal vor. Ist aus Benutzersicht ein Zugriff auf teilweise veraltete Zustände von Datenquellen akzeptierbar, so kann die Auftragsvergabe neben systemtechnischen Eigenschaften (Lastbalancierung, Kommunikationsaufwand) auch inhaltliche Aspekte berücksichtigen, wobei als Voraussetzung die Güte eines lokalen Zustands eines jeden Grid-Mitglieds zu quantifizieren ist.
Der Beitrag führt unterschiedliche Methoden ein, die eine Quantifizierung der Konsistenz eines Datenbankzustands für ein Grid-Mitglied hinsichtlich zeit- und datenbezogener Differenzen im Vergleich zu einer direkten Auswertung auf den Datenquellen erlauben. Es wird diskutiert, wie die auftretenden Differenzen klassifiziert und quantifiziert werden können, um dadurch dem Benutzer ein Gefühl für die Qualität des Ergebnisses und für das Zusammenpassen der Datenzustände unterschiedlichen Alters zu vermitteln. Für den Fall, dass eine Datenquelle zusätzlich eine Historisierung der Daten ermöglicht, werden Verfahren angegeben, die nach Vorgabe eines gewünschten Quantitätsmaßes die passenden Daten aufsuchen und miteinander kombinieren.

1 Einleitung

Die Bereitstellung eines einheitlichen Zugangs zu autonomen Datenquellen kombiniert mit einer effizienten Ausführung von Anfragen ist seit vielen Jahren Gegenstand von Forschung und kommerzieller Entwicklung. Aktuell verschärft sich das Problem durch die Verbreitung von ›*Grid-Computing*‹-organisierten Rechnerstrukturen ([FoKe99]), wobei der Ansatz des Grid-Computings im Wesentlichen darin besteht, dass eine Menge lokaler Rechnersysteme zu einem virtuellen Supercomputer zusammengeschlossen werden und anstehende Aufgaben zentral verwaltet und einzelnen Knoten zur Bearbeitung übergeben werden.

Der Technik des Grid-Computing ist aktuell sowohl allgemein als auch insbesondere aus Sicht der Datenbanksysteme eine große Aufmerksamkeit zu schenken. So drängt zum einen der Einsatz der Grid-Computing-Technologie aus dem (akademisch motivierten) Bereich des wissenschaftlichen Rechnens durch ausgereifte Basis-Technologie (the globus project; http://www.globus.org) immer mehr in den kommerziellen Bereich zur Durchführung umfassender Auswertung innerhalb einzelner Unternehmen (›*enterprise grid*‹). Zum anderen ist eine Transition von einer funktionszentrierten Ausrichtung (›*number crunching*‹) zu einer datenzentrierten Ausrichtung (›*data crunching*‹) auszumachen, die offensichtlicherweise eine umfassende Unterstützung durch eine entsprechende Datenbanktechnologie erfordert ([HoCa01], [RiKT02]). Erste Ansätze werden beispielsweise durch das ›*Database Access and Integration Services*‹-Forum (DAIS; http://www.cs.man.ac.uk/grid-db/) koordiniert.

Spannungsfeld der Grid-Datenbanksysteme

Die Anforderungen an Datenbanktechnologie für Grid-Computing sind vielfältig. So ist beispielsweise trotz der Verwendung offener WebService-Techniken ([Cera02], [Newc02]) im Rahmen der aktuellen ›*Open Grid Services Architecture*‹ (OGSA; http://www.globus.org/ogsa) eine Beschreibung von Datenbankdiensten für eine dynamische Nutzung nur unzulänglich möglich. Weiterhin sind die Möglichkeiten einer intelligenten Auftragsvergabe einer eingehenden Datenbankanfrage an ein oder mehrere Mitglieder des Grid-Datenbanksystems Gebiete, die weitreichende Forschungsmöglichkeiten eröffnen.

Mit Blick auf existierende Technologie existiert im Allgemeinen ein Spannungsfeld (Abbildung 1), in welchem Grid-Datenbanksysteme einzuordnen und entsprechende Technologien zu adaptieren sind.

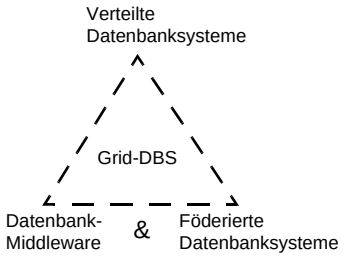


Abb. 1: Klassifikation von Grid-DBS

- *Verteilte Datenbankssysteme*
  Im Bereich der verteilten Datenbanksysteme ([ÖzVa99], [Dada96], [Rahm94]) werden einzelne Datenbankoperationen an partizipierende Systeme verteilt. Probleme der Lastbalancierung und Kostenabschätzung bei der Anfrageausführung (insbes. Kommunikationskosten) und Synchronisierung von Replikaten treten in Grid-Datenbanken erneut auf.
- *Datenbank-Middleware und Föderierte Datenbanksysteme*
  Die Bildung einer homogenen Datenbasis über einer Menge weitgehend autonomer Datenquellen auf technischer Ebene (insbes. DB-Middleware) und auf Ebene der Schema- und Datenintegration findet sich bei der Definition einer Grid-Datenbank in veränderterter bzw. spezialisierter Form wieder. DB-Middleware kann dabei als Basismechanismus dienen ([DiGe00]) und um entsprechende Funktionalität ergänzt werden.

Logisches Architekturmodell des verwendeten Grid-Datenbanksystems

Der vorliegende Beitrag greift die Problematik einer inhaltlich-motivierten Auftragsvergabe einer Datenbankanfrage an ein Mitglied der Grid-Computing organisierten Rechnerstruktur auf. Die Idee besteht darin, die Güte (im Sinne der Konsistenz) eines lokalen Da-

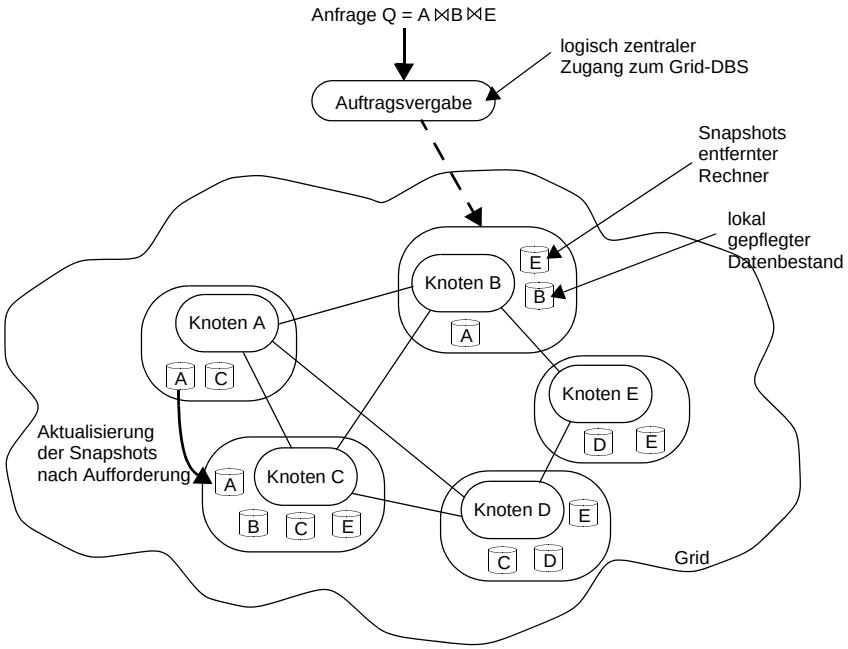


Abb. 2: Logische Architektur des diskutierten Grid-Datenbank-Szenarios

tenbestands quantitativ zu bewerten und damit eine Grundlage für eine geeignete Auswahl eines Rechners zur Auswertung der Anfrage zu schaffen. Dem Verfahren liegt folgendes logische Architekturmodell eines Grid-Datenbanksystems zu Grunde (Abbildung 2):

Das Grid besteht dabei aus einer Menge miteinander verbundener Rechner mit lokal autonomen Datenbanksystemen, die optional einen lokal gepflegten Datenbestand aufweisen können. Eine Anfrage an das System wird von der Auftragsverarbeitung an ein (zuvor ausgewähltes) Mitglied des Grids weitergeleitet, sodass das Prinzip der lokalen Aktualisierung und globalen Analyse von Datenbeständen zu Grunde liegt. Sofern nicht alle benötigten Daten für die Beantwortung einer Anfrage lokal zur Verfügung stehen, werden neue Snapshots bzw. aufgelaufene Änderungen zur Aktualisierung lokaler Snapshots angefordert und lokal zwischengespeichert.

Die Auftragsvergabe hängt dabei (im klassischen Ansatz) in erster Linie von systemtechnischen Parametern wie Auslastung der einzelnen Systeme und des zu erwartenden Kommunikationsaufwands ab. Kann bei einem lesenden Zugriff auf höchste Konsistenz verzichtet werden, indem zu Gunsten einer höheren Ausführungsgeschwindigkeit auf lokal zwischengespeicherte (und möglicherweise veraltete) Datenbestände zugegriffen werden darf, ist die Strategie der Auftragsvergabe durch inhaltliche Eigenschaften (Alter und Umfang der lokal verfügbaren Datenbestände) zu erweitern.

Voraussetzung dafür ist jedoch, dass dem Benutzer bei der Formulierung der Anfrage eine Spezifikation der *maximal zulässigen Inkonsistenz* ermöglicht wird. Des Weiteren muss im Zuge der Ermittlung des günstigsten Grid-Mitglieds lokal eine Berechnung der *zu erwar-*

tenden Inkonsistenz bei der Ausführung der Anfrage auf den lokal verfügbaren Datenbeständen gewährleistet werden. Genau dieser Problematik widmen sich die in diesem Bericht skizzierten Verfahren und Techniken.

Wissenschaftlicher Beitrag der Arbeit

Da die Snapshots, die zu unterschiedlichen Zeiten lokal an einem Grid-Mitglied abgelegt werden, nicht immer den aktuellsten Stand widerspiegeln und darüber hinaus die Daten selbst unterschiedliche Gültigkeitszeitpunkte besitzen, ergeben sich zeit- und datenbezogene Differenzen. Besonders deutlich wird dies beim Verbund, da hier Snapshots mit differierenden Zeitpunkten zusammengeführt werden. Im Rahmen des vorliegenden Beitrages werden die auftretenden Differenzen klassifiziert und quantifiziert sowie Verbundsemantiken diskutiert. Ergebnis ist die Einführung eines ›Fingerabdrucks‹ einer autonomen Datenquelle, an Hand dessen die Änderungscharakteristik abgeschätzt und für die Ermittlung der lokalen Inkonsistenz eines Datenbestands für einer Datenbankanfrage herangezogen werden kann.

Das folgende Kapitel diskutiert die Quantifizierung der Konsistenz aus rein zeitlicher Perspektive. Die orthogonale Richtung der Änderungsrate wird in Kapitel 3 aufgegriffen bevor in Kapitel 4 beide Aspekte zusammengeführt werden. Abgerundet wird der Beitrag durch eine kurze Zusammenfassung.

2 Auswahl von Datenobjekten aus zeitlicher Perspektive

Als erster Parameter für eine Ergebnisquantifizierung und damit für die Charakterisierung einer Griddatenquelle mittels eines Fingerabdrucks wird in diesem Kapitel ausschließlich die zeitliche Perspektive betrachtet ohne auf Zusammenhänge mit anderen Parametern wie dem Änderungsverhalten einzugehen. Dazu bringt Abschnitt 2.1 eine detailliertere Sicht und führt einige Termini ein. Der Quantifizierung der zeitlichen Differenzen widmet sich anschließend Abschnitt 2.2 bevor Abschnitt 2.3 auf eine Verfeinerung mittels Konsistenzbändern eingeht. Zum Abschluss des Kapitels erfolgt noch eine algorithmische Betrachtung der Problematik.

2.1 Problematik, Punktgraph und historischer Schnitt

Wie in Kapitel 1 angedeutet und in Abbildung 3 nochmals verdeutlicht wird ein neues oder geändertes Datenobjekt zunächst in eine operative Datenquelle D_o eingebracht, wodurch Transaktionszeit und Gültigkeitszeit ([SnAh85]) für das Datenobjekt festgelegt werden können. Unabhängig davon wird dann ein Snapshot, der dieses Objekt alleine oder eine Ansammlung von Datenobjekten enthält, von D_o an die konsumierende Datenquelle D_k gesandt, wodurch diesem Snapshot und damit den enthaltenen Datenobjekten ebenfalls eine Transaktions- und eine Gültigkeitszeit zugewiesen werden können. Damit existieren im Gesamtsystem vier unterschiedliche Zeitpunkte, wovon für die Auswertung jedoch nur die Gültigkeitszeit auf Seiten von D_o interessant ist. Besitzt das ursprüngliche geänderte

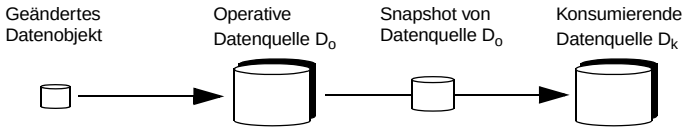


Abb. 3: Veranschaulichung von Datenobjekten und Snapshots im Gesamtsystem

Datenobjekt keine explizit ausgewiesene Gültigkeitszeit bzw. führt es selbst keine Zeitinformation mit, aus der sich die Gültigkeitszeit ergibt, so wird die Gültigkeitszeit gleich der Transaktionszeit von D_o gesetzt, falls auch diese nicht verfügbar ist, gleich der Transaktionszeit bei der Speicherung in D_k.

Datenquellen-Zeit-Diagramm (Punktgraph)

Im zeitlichen Verlauf werden pro operativer Datenquelle mehrere Snapshots erzeugt, sodass eine graphische Veranschaulichung mittels eines *Datenquellen-Zeit-Diagramms*, im Folgenden nur kurz *Punktgraph* genannt, erfolgen kann. Im Punktgraph wird für jede Datenquelle der Gültigkeitszeitpunkt des Snapshots in Form einer Matrix angetragen. So finden sich im Beispiel der Abbildung 4 für die erste Datenquelle ein Snapshot, für die zweite zwei und für die letzte drei, wobei S_n^3 erst später als der aktuelle Zeitpunkt t_{NOW} gültig wird.

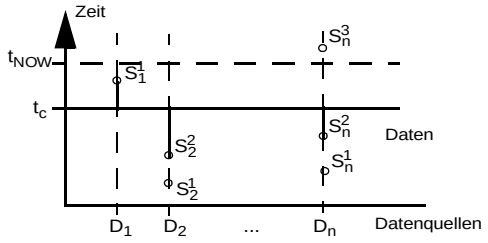
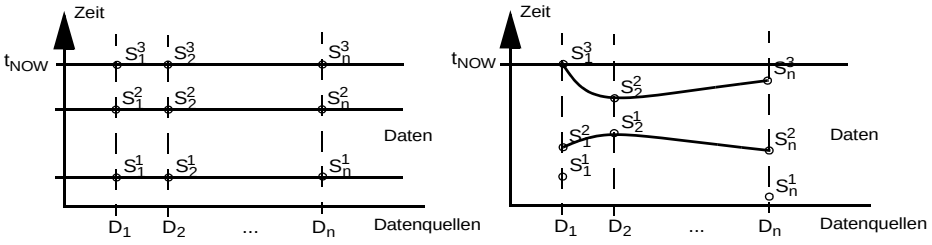


Abb. 4: Beispiel eines Punktgraphen von Gültigkeitszeiten für verschiedene Datenquellen

Historischer Schnitt

Üblicherweise werden die aktuell gültigen Snapshots miteinander kombiniert ohne den Zeitpunkt, ab wann sie gültig geworden sind, zu berücksichtigen. Im Punktgraph sind das diejenigen Snapshots, die auf der Zeitachse von der Vergangenheit her gesehen am nähesten bei t_{NOW} liegen. Im Idealfall besitzen alle Snapshots denselben Gültigkeitszeitpunkt, was sich graphisch durch die zeitlich jüngste horizontale Linie widerspiegelt (Abbildung 5 links). Diese sowie auch jede weitere horziontale Linie wird *historischer Schnitt* t_c genannt.

Im Normalfall werden jedoch die Snapshots zu unterschiedlichen Zeitpunkten gültig, weshalb das Verbinden der jüngsten Snapshots dann nicht mehr zu einer horizontalen Linie (Abbildung 5 rechts) führt. Während eine horizontale Linie ein zeitlich optimales Zusammenpassen der Snapshots reflektiert, ist dies bei einer beliebigen Kurve nicht mehr der Fall. Vielmehr gilt es dann bei der Bestimmung eines historischen Schnittes, um den sich die Snapshots gruppieren, zum einen dem Idealfall einer horizontalen Linie sehr nahe zu kommen, gleichzeitig aber auch möglichst aktuelle Snapshots zu selektieren, um damit einen weitestgehend jungen historischen Schnitt zu erhalten. Die Bewertung und damit die Quantifizierung der Qualität des Ergebnisses der gewählten Snapshots aus zeitlicher Sicht



(a) Idealfall (b) Realfall

Abb. 5: Idealfall versus Realfall bei der Auswahl von Snapshots

wird im folgenden Abschnitt näher betrachtet, bevor sich die weiteren Abschnitte mit der Bestimmung von Snapshots, die dem Idealfall möglichst nahe kommen, sowie der Verbundsemantik und dem damit erzeugten Fehlern widmen.

2.2 Quantifizierung der Qualität mittels eines Inkonsistenzmaßes

Wird im Datenbankbereich von Konsistenz gesprochen, so bezieht sich dies in der Regel auf die Eigenschaften einer Transaktion im Rahmen des ACID-Prinzips ([ElNa00], [Date99]). Bei der vorliegenden Architektur wird davon ausgegangen, dass auf die Datenquellen nur lesend zugegriffen wird und Datenänderungen lokal durchgeführt werden. Deshalb hat jede Datenquelle für sich die *Datenkonsistenz* aufrecht zu erhalten.

Als weitere Konsistenzart kann im weiteren Sinne die *Zeitkonsistenz* angeführt werden. Diese setzt die Gültigkeitszeitpunkte von je einem Snapshot der verschiedenen Datenquellen in Bezug zueinander, indem der zeitliche Abstand der Snapshots zum historischen Schnitt t_c aufsummiert wird:

$$I = \sum_{i=1}^{k} \left| (\text{time}(S_i^j) - t_c) \right|$$

Für den Idealfall ergibt sich $I = 0$ und je weiter die einzelnen Snapshots von t_c entfernt sind, desto größer wird I. Deshalb ergibt sich die Konsistenz als Qualitätsmaß in Form einer Inkonsistenz.

Treten alle Snapshots mit der gleichen Wahscheinlichkeit auf ($p_1 = p_2 = \ldots = p_k = \frac{1}{k}$), so ergibt sich die mittlere Abweichung, gleichzeitig Erwartungswert bei einer Gleichverteilung der Wahrscheinlichkeiten, zu $\mu = \frac{I}{k}$. Die Standardabweichung berechnet sich damit wie folgt:

$$\sigma^2 = \frac{\sum_{i=1}^{k} (\text{time}(S_i^j) - t_c)^2}{k} - \mu^2$$

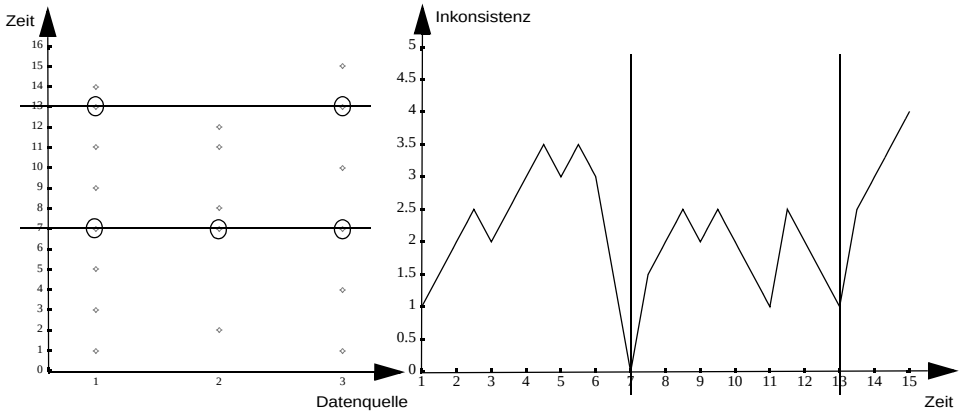


Abb. 6: Beispiel für die Wahl historischer Schnitte und der sich ergebenden Inkonsistenzen

Ein Beispiel zeigt Abbildung 6. Im linken Teil sind für drei Datenquellen Snapshots mit ihren Gültigkeitszeiten zu sehen. Wird zum Zeitpunkt 13 der historische Schnitt genommen und die nächstliegenden Snaphots gewählt, so ergibt sich eine Inkonsistenz vom Wert 1, die in der rechten Graphik angetragen wird. Dieselbe Vorgehensweise wird für jeden Zeitpunkt der linken Graphik angewandt, wodurch sich die Inkonsistenzkurve der rechten Graphik ergibt. Zu erkennen ist eine im Umfeld des Grid-Computing enstehende Problematik beim Vergleich der Inkonsistenzen zum Zeitpunkt 7 und 13: Zum Zeitpunkt 13 ist die Inkonsistenz zwar größer, aber der Schnitt ist zeitlich jünger, während zum Zeitpunkt 7 optimale Konsistenz gegeben ist, die Snapshots aber weitaus älter sind.

2.3 Verfeinerung durch Konsistenzbänder

Werden die gewählten Snapshots mit einem normalen inneren Verbund verknüpft, so werden die zeitlichen Differenzen zwischen den Snapshots und dem historischen Schnitt nicht berücksichtigt. Vielmehr bedeutet diese Verbundart, dass alle Snapshots untereinander als (zeitlich ausreichend) nahe zusammenliegend und damit untereinander als konsistent angesehen werden. Beim Verbund reflektiert werden kann die zeitliche Differenz durch den äußeren Verbund, der auch in föderativen Datenbanksystemen das übliche Mittel zur Integration darstellt ([Conr97]). Allerdings können bei ausschließlicher Anwendung Ergebnisse produziert werden, die keine Aussagekraft mehr besitzen und damit nicht verwendbar sind. Das Pendant dazu stellt die Nicht-Verwendung dar, was zu denselben Ergebnissen wie bei Verwendung des inneren Verbundes führen kann. Deshalb erscheint eine Kombination aus beiden am sinnvollsten.

Der hier vorgestellte Ansatz legt zur Bestimmung der Anwendung des inneren und äußeren Verbundes um den historischen Schnitt ein inneres und äußeres Konsistenzband, wie es in Abbildung 7 zu sehen ist. Das *äußere Konsistenzband* KB_A gibt den maximal zulässigen Abstand eines Snapshots vom historischen Schnitt an. Alle Snapshots, die innerhalb

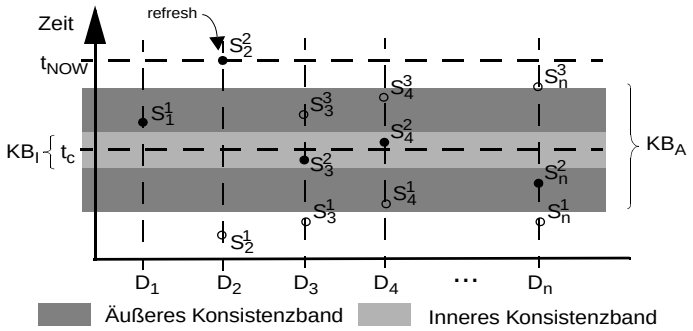


Abb. 7: Beispiel eines inneren und äußeren Konsistenzbandes

des KB_A liegen, werden mittels eines äußeren Verbundes verknüpft. Falls für eine Datenquelle kein Snaphost innerhalb des äußeren Konsistenzbandes liegt (bspw. S_2^1 in Abbildung 7), so muss eine Sonderbehandlung durchgeführt werden. Beispielsweise ist eine Erneuerung des Snapshots mit aktuellen Daten (refresh, Abbildung 7) denkbar.

Liegen die Snapshots innerhalb des *inneren Konsistenzbandes* KB_I, so werden diese mittels eines inneren Verbundes zusammengeführt. Durch KB_I wird der Tatsache Rechnung getragen, dass Snapshots als zeitlich konsistent angesehen werden, auch wenn eine kleine, aber aus globaler Sicht vernachlässigbare Abweichung der Gültigkeitszeiten vorhanden ist. In Abbildung 7 ist dies beipielsweise für S_3^2 und S_4^2 der Fall.

2.4 Algorithmus zur Bestimmung des historischen Schnittes

Einen einfachen, heuristischen Algorithmus zur Bestimmung des historischen Schnittes und damit zur Auswahl der Snapshots unter Berücksichtigung von innerem und äußerem Konsistenzband zeigt Anhang A1. Bezüglich des Beispiels aus Abbildung 6 würde das Ergebnis zum Zeitpunkt 13 erzeugt werden bei Annahme eines äußeren Konsistenzbandes von 8 Zeiteinheiten symmetrisch um t_c. Weiterführende Algorithmen werden in [ScLe02] diskutiert.

3 Datenänderungsrate

Nachdem in Kapitel 2 eine ausschließliche Fokussierung des Problembereichs aus zeitlicher Perspektive erfolgt, wird in diesem Kapitel orthogonal dazu die Änderungsrate betrachtet. Im Folgenden wird deshalb zunächst eine Klassifizierung der Änderungsrate vorgenommen (Abschnitt 3.1) bevor anschließend auf die Berechnung der Änderungsrate eingegangen wird (Abschnitt 3.2).

3.1 Klassifizierung der Änderungsrate

Die Änderungsrate kann prinzipiell von einem Snapshot zum anderen bzw. für eine Datenquelle insgesamt konstant sein, sich nach einer Verteilung oder Funktion richten oder völlig willkürlich sein. Zur weiteren, in diesem Beitrag verwendeten Klassifizierung wird angelehnt an [TCG+93] zunächst eine Einteilung der Datenquellen in zwei Klassen vorgenommen:

- *Zustandsreflektierende Quelle*
  Bei diesem Typ wird ausschließlich der Wert eines Datenobjektes zum Snapshotzeitpunkt festgehalten, wie es in Abbildung 8 angedeutet ist. An die konsumierende Datenquelle wird das Tripel *(ObjektID, Zeitpunkt, Wert)* weitergegeben.

- *Historienbewahrende Quelle*
  Das Charakteristikum dieses Quellentyps ist, dass der Werteverlauf über die Zeit aufgehoben wird, sodass für ein Datenobjekt mehre Zeitpunkte mit einem Wert existieren (Abbildung 8). Damit besteht eine Snapshot für ein Objekt aus dem Tripel *(ObjektID, (Zeitpunkt, Wert)*)*.

Eine Verfeinerung dieser Basisklassifikation der Datenquellen kann durch Betrachtung des Änderungsintervalls der Datenobjekte erreicht werden. Bei einem regulären Änderungsintervall ist der Abstand zwischen zwei Änderungen immer konstant, bei einem irregulären Änderungsintervall kann er variieren.

Ferner ist zu unterscheiden, ob es sich bei den Änderungen um existenzielle oder wertebasierte Änderungen handelt. Unter einer existenziellen Änderung wird das Erzeugen oder Löschen eines Objektes verstanden, während wertebasierte Änderungen ein bestehendes Objekt modifizieren.

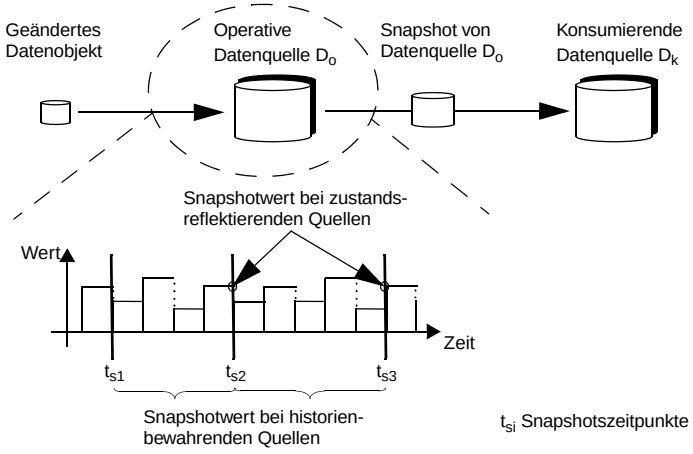


Abb. 8: Klassifizierung der Datenquellen

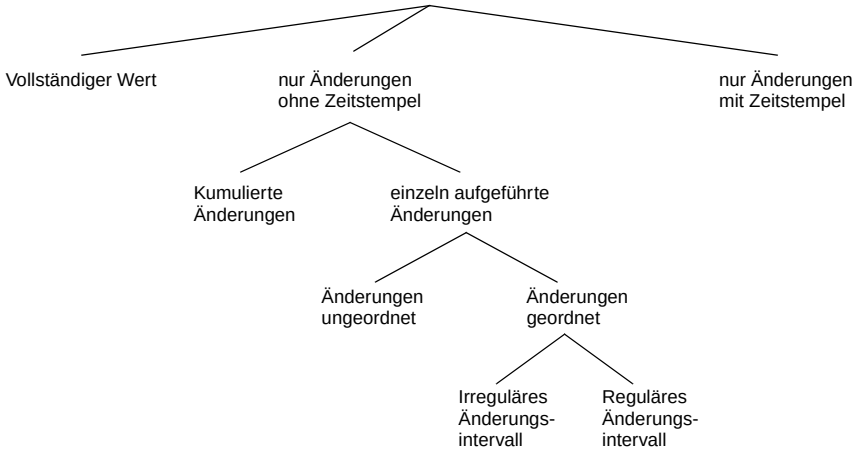


Abb. 9: Arten der Progapierung von Änderungen

Die Modifikationen selbst können in unterschiedlichen Formen propagiert werden, unabhängig davon, ob es sich um existenzielle oder wertebasierte Änderungen handelt. Eine Klassifizierung ist Abbildung 9 zu entnehmen und wird im folgenden erläutert:

- *Vollständiger Wert*
  Zu den einzelnen Zeitpunkten wird immer der jeweils gültige Wert bereitgestellt.
- *Nur Änderungen ohne Zeitstempel*
  Hierbei sind die Modifkationsoperationen, seien sie existenziell im Sinne eines Einfügens (Insert) oder Löschens (Delete) oder seien sie wertebasiert im Sinne einer wertmäßigen Änderung (Update), ohne den Zeitpunkt ihrer Änderung aufgeführt. Eine weitere Unterscheidung kann dahingehend getroffen werden, ob Operationen auf dem selben Objekt zu einer absoluten Änderung zusammengefasst werden können (kumulierte Änderung) oder ob sie einzeln aufgeführt sind. Im letzten Fall können die Änderungen ungeordnet oder geordnet vorliegen. Ist die Datenquelle durch ein reguläres Änderungsintervall charakterisiert, so kann darüber hinaus bei Vorliegen einer Ordnung unter den Änderungen zusammen mit einem Anfangs- oder Endzeitpunkt jede Änderung einem exakten Zeitpunkt zugeordnet werden.
- *Nur Änderungen mit Zeitstempel*
  Dieser Fall ist ähnlich gelagert zum vorhergehenden mit dem Unterschied, dass jede Änderung mit einem Zeitstempel (Gültigkeitszeitpunkt) versehen ist. Dadurch entfällt die vormals durchgeführte weitere Klassifikation.

Werden die durchgeführten Klassifkationen zusammengeführt, so ergibt sich, dass alle Kombinationen zulässig und sinnvoll sind. Anzumerken ist, dass für Objekte, die zum Abzugszeitpunkt nicht mehr existieren, noch wertebasierte Änderungsoperationen vorliegen können. Diese werden einfach nicht weiter beachtet.

Verbund von Snapshots

Werden Snapshots gleichen oder unterschiedlichen Typs miteinander verbunden, so sollte der Abgeschlossenheit wegen das Ergebnis von einem der angeführten Typen sein. Dies ist, wie Tabelle 3.1 zeigt, erfüllt. Damit ist neben der Kommutativität beim Verbund von Relationen auch deren Assoziativität bzgl. der Typen indirekt nachgewiesen.

	historienbewahrend	**zustandsreflektierend**
historienbewahrend	historienbewahrend	historienbewahrend
zustandsreflektierend	historienbewahrend	zustandsreflektierend

Tab. 3.1: Typen nach einem Verbund

Neben den entstehenden Datentypen ist das erzeugte Zeitraster interessant, was Tabelle 3.2 aufführt. Die Fälle, in den zwei Quellen gleichen Typs verbunden werden, und die gemäß der Tabelle 3.2 eine Mischung beider Raster erfordern, ist beispielhaft in Abbildung 10 skizziert.

	historienbewahrend	**zustandsreflektierend**
historienbewahrend	Mischung beider Raster	Raster der historien-bewahrenden Quelle
zustandsreflektierend	Raster der historien-bewahrenden Quelle	Mischung beider Raster

Tab. 3.2: Zeitraster nach einem Verbund

3.2 Berechnung der Änderungsrate

Auf Seite der konsumierenden Datenquelle treffen entsprechend der vorgenommenen Klassifikation aus Abschnitt 3.1 entweder einzelne Snapshots oder nur die Änderungsoperationen ein, aus denen für eine Charakterisierung der Datenquelle eine Änderungsrate zu berechnen ist.

Der Prozentsatz der Einfüge-, Lösch- oder Änderungsoperationen im Verhältnis zur Ausgangsgröße zwischen zwei Zeitpunkten wird Änderungsrate genannt, wobei es je eine für existenzielle und eine für wertebasierte gibt. Wird der zeitliche Abstand normiert, so kann

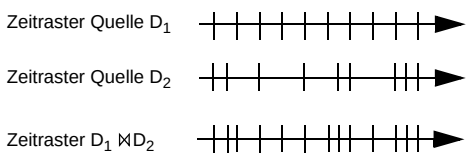


Abb. 10: Entstehendes Zeitraster beim Mischen

die Änderungsrate konstant sein, sich nach einer Funktion oder Verteilung richten oder völlig irregulär sein. Im Rahmen des Beitrags wird von einer konstanten Änderungsrate ausgegangen, die entweder von der Datenquelle geliefert wird oder wie folgt berechnet werden kann:

- *Vorliegen von Änderungsdeltas*
  In diesem Fall liegt die Zahl der Änderungsoperationen n_i^d innerhalb des Intervalls Δt_i explizit vor und die Änderungsrate $\Delta d_i^{\Delta t}$ kann direkt berechnet werden:

$$\Delta d_i^{\Delta t} = \frac{n_i^d}{\Delta t_i}$$

- *Vorliegen von Snapshots*
  Seien für eine Datenquelle i zwei Snapshots zum Zeitpunkt t_i^j und t_i^k mit o.B.d.A. $t_i^j \leq t_i^k$ gegeben und sei ferner $PKA(S_i^l)$ eine Funktion, die die Instanzen der Primärschlüsselattribute des Snapshots der Quelle i zum Zeitpunkt l zurückgibt, so gilt für die jeweilige relative Anzahl an
  - Einfügungen: $ins_i^{jk} = PKA(S_i^k) - PKA(S_i^j)$
  - Löschungen: $del_i^{jk} = PKA(S_i^j) - PKA(S_i^k)$
  - Änderungen: $upd_i^{jk} = PKA(S_i^j - S_i^k) \cap PKA(S_i^k - S_i^j)$

  Somit ergibt sich die Änderungsrate $\Delta d_i^{\Delta t}$ mit $\Delta t_i = t_i^k - t_i^j$ zu

$$\Delta d_i^{\Delta t} = \frac{ins_i^{jk} + del_i^{jk} + upd_i^{jk}}{\Delta t_i}$$

Anzumerken ist, dass durch die angegebenen Operationen im zweiten Fall Kompensationsoperationen (Einfügen und Löschen desselben Objektes) oder Mehrfachoperationen auf einem Objekt (Ändern eines Attributwertes) nicht bemerkt werden, wodurch die Änderungsrate eine andere ist im Vergleich zum ersten Fall, sofern dort keine kompensierenden Operationen verwendet werden. Zur Erreichung derselben Änderungsrate müssen im ersten Fall absolute Änderungsdeltas erzeugt werden. Um schließlich in beiden Fällen eine konstante Änderungsrate Δd_i pro Zeiteinheit zu erreichen, wird von einem Mittelwert ausgegangen, der nach einer Einschwingphase des Gesamtsystems erreicht werden kann. Dieser Wert kann sowohl für jede Operation als auch nach Zusammenfassung als Gesamtänderungsrate angegeben werden, wobei eine Gewichtung der einzelnen Änderungswerte denkbar ist.

Eine Verfeinerung der bisherigen Diskussion erfolgt durch Betrachtung der Zeiteinheiten zwischen zwei Snapshots. Obwohl die zeitliche Differenz zwischen zwei Snapshots beliebig sein kann, wird die Änderungsrate bisher nur auf eine Zeiteinheit bezogen. Wird ferner berücksichtigt, dass bereits geänderte Objekte oder Attribute wieder geändert werden können, ein Teil zum ersten Mal geändert wird und der Rest unverändert bleibt, so ergeben sich die Anzahl der geänderten Tupel Δn_i bei t Zeiteinheit mit k als Zahl der Ausgangselemente und Δd_i der festen Änderungsrate zwischen zwei Zeiteinheiten zu $\Delta n_i = k - (k \bullet (1 - \Delta d_i)^t)$. Angenommen wird, dass im Mittel die Zahl der Tupel erhalten bleibt.

4 Kombination von zeitlicher Perspektive und Änderungsrate

Während sich das Kapitel 2 der Auswahl von Datenobjekten ausschließlich aus zeitlicher Perspektive widmet und im letzten Kapitel alleine die Bestimmung der Änderungsrate diskutiert wird, erfolgt jetzt eine Kombination von beiden Aspekten zum gewünschten Fingerabdruck einer Datenquelle. Dazu führt der Abschnitt 4.1 den Wiederverwendungsgrad ein. Der Quantifizierung der Inkonsistenz widmet sich anschließend Abschnitt 4.2 bevor im letzten Abschnitt auf einen Algorithmus zur Bestimmung des historischen Schnittes eingegangen wird.

4.1 Wiederverwendungsgrad ρ

Um das Alter eines Snapshots mit der Änderungsrate zusammenzuführen, wird die Wiederverwendbarkeit desselben betrachtet, die sich aus der Änderungsrate ergibt. Denn ist die Änderungsrate gering, so können auch alte Snapshots verwendet werden, ohne die Qualität des Endergebnisses zu stark zu beeinträchtigen, da der Unterschied zwischen alten und neuen Snapshots gering ist. Bei hoher Änderungsrate verhält es sich genau umgekehrt, da ein nicht mehr ganz aktueller Snapshot das Ergebnis stark beeinträchtigt. Damit kann der *Wiederverwendungsgrad* ρ in Abhängigkeit von der Änderungsrate Δd_i jedoch noch ohne Berücksichtigung des Alters wie folgt eingeführt werden: $\rho(\Delta d_i) = -\Delta d_i + 1$.

Bei der Erweiterung dieser Basisformel um den angesprochenen temporalen Aspekt muss beachtet werden, dass unabhängig von der Änderungsrate der *absolute Wiederverwendungsgrad* bei Snapshots, die zum aktuellen Zeitpunkt t_{NOW} erstellt wurden, 100% beträgt. Während t_{NOW} die eine zeitliche Grenze bildet, ist das Alter (also die Gültigkeitszeit) des ältesten Snapshots (t_S) die andere Grenze. Damit ergibt sich folgende Funktion, wofür einige Beispielgraphen in Abbildung 11 zu sehen sind. Zu erkennen ist, dass die Kurven symmetrisch zu ρ(0.5) sind:

$$0 < \Delta d_i \le 0,5 : \rho(t) = \frac{-1}{\left|t - (t_{NOW} - t_S) - 1\right|^{\frac{1}{2\Delta d_i} - 1}} \bullet \left(\frac{1}{t_{NOW} - t_S} \bullet (t - (t_{NOW} - t_S)) + 1\right) + 1$$

$$0,5 \le \Delta d_i < 1 : \rho(t) = \frac{1}{(t+1)^{\frac{-1}{2(\Delta d_i - 1)} - 1}} \bullet \left(\frac{-1}{t_{NOW} - t_S} \bullet t + 1\right)$$

Offensichtlich können an Stelle von t_{NOW} und t_S auch andere Zeitpunkte t_E und t_A ($t_E \ge t_A$) als End- und Anfangszeitpunkt definiert werden, wodurch ein *relativer Wiederverwendungsgrad* gegeben ist. Falls $t_E = t_c$ ist, so ist der Wiederverwendungsgrad eines Snaps-

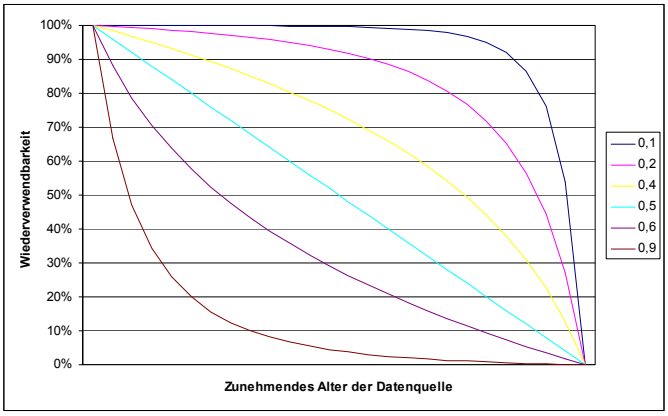


Abb. 11: Zusammenhang zwischen Wiederverwendungsgrad und Alter

hots, der jünger als t_E ist, 100%. Für Werte, die älter als t_c sind, gibt es somit zwei unterschiedliche Werte, einen absoluten und einen relativen Wiederverwendungsgrad, auf die im folgenden Abschnitt näher eingegangen wird.

4.2 Maß für die Inkonsistenz

Die Aufstellung eines Inkonsistenzmaßes analog zu Abschnitt 2.2 muss sowohl die zeitliche Dimension, als auch die Änderungsrate respektive den Wiederverwendungsgrad berücksichtigen. Weiterhin ist zu differenzieren, ob ein Konsistenzmaß bezüglich des aktuellen Zeitpunktes t_{NOW} (*Aktualitätsbezogenes Konsistenzmaß*, I_A) oder des Zeitpunktes des historischen Schnitts t_c (*Historienbezogenes Konsistenzmaß*, I_H) definiert wird.

Aktualitätsbezogenes Konsistenzmaß I_A

Um ein absolutes Maß für die Qualität des Ergebnisses bezogen auf den aktuellen Zeitpunkt t_{NOW} zu erhalten, ist der Endzeitpunkt t_E des Wiederverwendungsgrades gleich t_{NOW} zu setzen. Dadurch wird der Wiederverwendungsgrad jedes Snapshots in ein absolutes Verhältnis zum aktuellen Zeitpunkt gesetzt, wodurch eine Bewertung der verwendeten Snapshots bezogen auf t_{NOW} erfolgen kann.

Historienbezogenes Konsistenzmaß I_H

Durch das Setzen von t_E auf den Zeitpunkt des historischen Schnittes t_c werden die Snapshots in Bezug zu t_c gesetzt, wodurch eine Bewertung in Form eines Konsistenzmaßes bezogen auf den Abstand zu t_c erfolgt. Indirekt ist damit ein Maß gegeben, wie gut die einzelnen Snapshots zusammenpassen. So können, um nur ein Beispiel zu nennen, alle ausgewählten historischen Schnitte denselben Zeitpunkt besitzen wie t_c, wodurch $I_H = 0$ ist, jedoch könnte $I_A \neq 0$ sein, sofern $t_c \neq t_{NOW}$ ist.

Berechnung des Inkonsistenzmaßes I

Der Unterschied zwischen dem aktualitätsbezogenem und dem historienbezogenem Inkonsistenzmaß liegt nur in der Festlegung des Endzeitpunktes. Des Weiteren muss die Formel für das Inkonsistenzmaß berücksichtigen, dass bei 100-prozentiger Wiederverwendbarkeit genauso wie bei einem zeitlichen Abstand von Null der Snapshots zu t_c oder t_{NOW} minimale Inkonsistenz erzielt wird, wobei als Minimum Null festgelegt wird, da negative Werte keinen Sinn ergeben. Damit ergibt sich der Zusammenhang zwischen dem Wiederverwendungsgrad und der Zeit für eine Datenquelle i und dem zeitlichen Abstand Δt_i zwischen historischem Schnitt und Zeitpunkt des Snapshots time(S_i^j) wie folgt:

$$I_i(time(S_i^j)) = \Delta t_i \bullet (1 - \rho_i(time(S_i^j)))$$

Graphisch kann dies im Punktgraph dadurch veranschaulicht werden, dass die Linie zwischen einem Snapshot und t_c nicht mehr die Breite 0, sondern die Breite 1-ρ_i(time(S_i^j)) hat. Die Fläche spiegelt dann die Inkonsistenz wider, wie es in Abbildung 12 als Erweiterung von Abbildung 4 zu sehen ist.

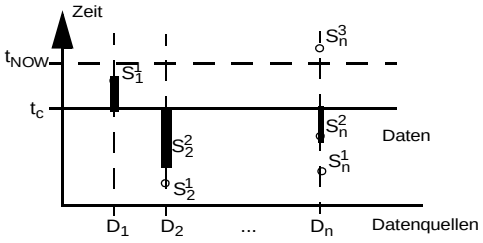


Abb. 12: Beispiel eines Punktgraphen von Gültigkeitszeiten für verschiedene Datenquellen

Eine Gewichtung der eingehenden Faktoren Änderungsrate und Wiederverwendungsgrad ist denkbar. Des Weiteren seien zur logischen Verifizierung der Formel die Fälle betrachtet, in denen die Zeit den Wert 0 oder der Wiederverwendungsgrad den Wert 0 oder 100 annimmt.

- *$\rho = 100, \Delta t = 0$ und $\rho \neq 0, \Delta t = 0$*
  Da die Snapshotzeit mit dem Zeitpunkt des historischen Schnitts übereinstimmt, ist unabhängig von dem Wiederverwendungsgrad der Snapshot bezüglich des historischen Schnittes immer der aktuellste und das Inkonsistenzmaß somit Null.
- *$\rho = 0, \Delta t \neq 0$*
  Der Snapshot ist zwar älter als der historische Schnitt, doch da keine Änderungen vorliegen, wäre der Snapshot auch zum Zeitpunkt des historischen Schnittes derselbe. Damit ist das Konsistenzmaß von Null gerechtfertigt.

Sind wiederum n Datenquellen beteiligt, so ergibt sich das Gesamtinkonsistenzmaß durch Summation der Einzelkonsistenzmaße, wobei die $S_k^{j_k}$ die gewählten Snapshots sind und durch α_k eine Gewichtung der Einzelkonsistenzmaße vorgenommen werden kann:

$$I(time(S_1^{j_1}), \ldots, time(S_k^{j_k})) = \sum_{k=1}^{n} (\alpha_k \bullet I_k(time(S_k^{j_k}))) = \sum_{k=1}^{n} (\alpha_k \bullet \Delta t_k \bullet (1 - \rho_k(time(S_k^{j_k}))))$$

4.3 Algorithmus zur Bestimmung des historischen Schnittes

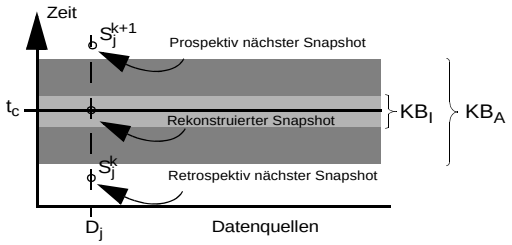


Abb. 13: Veranschaulichung der Snapshotsauswahl

Der in Abschnitt 2.4 vorgestellte Algorithmus zur Auswahl eines historischen Schnittes und damit von Snapshots muss so erweitert werden, dass der in den letzten beiden Kapiteln diskutierte Datenaspekt Berücksichtigung findet. Dazu wird zum ersten die einfache Inkonsistenzformel durch die in Abschnitt 4.2 eingeführte ersetzt. Des Weiteren ermöglicht das Wissen um die Art der Datenquelle eine weiterführende Auswahlmöglichkeit bei der Behandlung von nicht-existierenden Snapshots innerhalb des äußeren Konsistenzbandes. Während bei dem in Abschnitt 2.4 diskutierten Algorithmus nur die Möglichkeit besteht, einen neuen Snapshot von der operativen Datenquelle anzufordern und den Auswahlprozess von neuem zu starten, kann jetzt ein Snapshot zum Zeitpunkt t_c rekonstruiert werden. Dabei kann der exakt vorliegende Wert verwendet werden, sofern die Datenquelle die Änderungen mit einem Zeitstempel versieht. Ist dies nicht gegeben, so kann über die Zahl der Änderung und das Wissen des Änderungsintervalls oder durch Errechnung eines Wertes aus dem des retrospektiv und prospektiv nächsten Snapshots bezüglich t_c ebenfalls ein Snapshot rekonstruiert werden (Abbildung 13).

Den erweiterten Algorithmus zur Bestimmung des historischen Schnittes zeigt Anhang A2, wobei zur leichteren Lesbarkeit eine Vereinfachung dahingehend vorgenommen wird, dass auf die Aktualisierungsoperation verzichtet wird.

5 Zusammenfassung

Beim Konzept des Grid-Computing hält ein Mitglied des Grids neben lokal gepflegten Datenbeständen auch Snapshots der Datenbestände von anderen Mitgliedern aus dem Grid. Im Rahmen der Auftragsvergabe wird jedes Mitglied über seinen vorliegenden Datenbestand befragt, wobei neben systemtechnischen Eigenschaften die Güte des lokalen Datenbestandes die Auftragsvergabe beinflussen kann. Der Quantifizierung der Güte widmet sich der vorliegende Beitrag, wobei nach einer kurzen Einleitung (Kapitel 1) in drei Schritten vorgegangen wird: Nach einer eingehenden Betrachtung der Güte allein aus zeitlicher Perspektive (Kapitel 2) wird die dazu orthogonal liegende Änderungsrate (Kapitel 3) diskutiert. In Kapitel 4 schließlich werden beide Aspekte in Form eines Fingerabdrucks zusammengeführt, der die Güte der Datenquelle widerspiegelt.

Die vorgestellten Ideen werden im Rahmen des Forschungsprojektes SCINTRA (Semi-Consistent Timeorientied Replication Aspect) realisiert, wobei relationale Datenbanksysteme die Mitglieder des Grids repräsentieren.

Anhang

A 1: Algorithmus SimpleDeterminationHistoricCut

Algorithmus zur Bestimmung von historischem Schnitt und zugehörigen Snapshots

```
Algorithmus SimpleDeterminationHistoricCut

Input:      D       // Menge von Datenquellen D_i jede mit Snapshots S_i^j
            KB_A    // Äußeres Konsistenzband
Output:     t_c     // Historischer Schnitt
            S       // Menge an Snapshots


Begin
    Do
        allSnapshotsNotInKBA = true
        t_new = ∞
        I_new = ∞
        S_new = [∞,∞,...,∞]
        Do
            t_c = -∞
            t_old = t_new
            d_old = d_new
            S_old = S_new
            // finde nächsten Snapshot: dessen Zeitpunkt ist
            // der nächste historische Schnitt
            Foreach D_i in D
                Foreach S_i^j in D_i
                    If ( time( S_i^j ) < t_old And time( S_i^j ) > t_c )
                        t_c = time( S_i^j )
                    End If
                End foReach
            End Foreach
            // berechne für jede Datenquelle den zu t_new nächstliegenden Snapshot
            Foreach D_i In D
                S_i^s = ∞
                Foreach S_i^j In D_i
                    If ( |time( S_i^j ) - t_c| ≤ |time( S_i^s ) - t_c| && time( S_i^s ) < t_NOW )
                        S_i^s = S_i^j
                    End If
                End Foreach
                t_new = t_c
                S_new[i] = S_i^s
                I_new = I( t_new, S_new ) // Funktion zur Berechnung der neuen Inkonsistenz
            End Foreach
        While ( I_new < I_old )
            // überprüfe, ob alle Snapshots innerhalb von KB_A liegen
        Foreach S_i^j In  S_old
            allSnapshotsNotInKBA = false
            If ( | time( S_i^j ) - t_c| > ( t_c + KB_A/2 ) )
                refresh( D_i )
                allSnapshotsNotInKBA = true
            End If
        End Foreach
    While ( allSnapshotsNotInKBA )
    Return ( t_old, S_old )
End
```

A2: Algorithmus DeterminationHistoricCut

Erweiterter Algorithmus zur Bestimmung von historischem Schnitt und zugehörigen Snapshots

```
Algorithmus DeterminationHistoricCut

Input:      D       // Menge von Datenquellen D_i jede mit Snapshots S_i^j
            KB_A    // Äußeres Konsistenzband
Output:     t_c     // Historischer Schnitt
            S       // Menge an Snapshots


Begin
    t_new = ∞
    I_new = ∞
    S_new = [∞,∞,...,∞]
    Do
        t_c = -∞
        t_old = t_new
        d_old = d_new
        S_old = S_new
        // finde nächsten Snapshot: dessen Zeitpunkt ist
        // der nächste historische Schnitt
        Foreach D_i In D
            Foreach S_i^j In D_i
                If ( time( S_i^j ) < t_old And time( S_i^j ) > t_c )
                    t_c = time( S_i^j )
                End If
            End Foreach
        End Foreach
        // berechne für jede Datenquelle den zu t_new nächstliegenden Snapshot
        Foreach D_i In D
            S_i^s = ∞
            Foreach S_i^j In D_i
                If ( |time( S_i^j ) - t_c| ≤ |time( S_i^s ) - t_c| && time( S_i^s ) < t_NOW ) )
                    S_i^s = S_i^j
                End If
            End Foreach
            t_new = t_c
            S_new[i] = S_i^s
            I_new = I( t_new, S_new ) // Funktion zur Berechnung der neuen Inkonsistenz
        End Foreach
        While ( I_new < I_old )
            // überprüfe, ob alle Snapshots innerhalb von KB_A liegen
        Foreach S_i^j In  S_old
            If ( | time( S_i^j ) - t_c| > ( t_c + KB_A/2 ) )
                S_i^j = computeSnapshot ( D_i, t_c ) // Berechnung des Snapshots bei t_c
            End If
        End Foreach
    Return ( t_old, S_old )
End
```

Literatur

Cera02 Cerami, E.: *Web Services Essentials.* O'Reilly, Cambridge u.a., 2002

Conr97 Conrad, S.: *Föderierte Datenbanksysteme*. Springer-Verlag, Berlin u.a., 1997

Dada96 Dadam, P: *Verteilte Datenbanken und Client-Server-Systeme. Grundlagen, Konzepte und Realisierungsformen*. Springer-Verlag, Berlin u.a., 1996

Date99 Date, C.J.: *An Introduction to Database Systems*. Addison-Wesley, Reading (MA) u.a., 1999

DiGe00 Dittrich, K.R.; Geppert, A. (Hrsg.): *Component Database Systems*. Morgan-Kaufmann-Verlag, San Francisco (CA) u.a., 2000

ElNa00 Elmasri, R.A.; Navathe, S.B.: *Fundamentals of Database Systems*. Addison-Wesley, Reading (MA) u.a., 2000

FoKe99 Foster, T.; Kesselman, C. (Hrsg.): *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann-Verlag, San Francisco (CA) u.a., 1999

HoCa01 Hoschek, W.; McCance, G.: Grid Enabled Relational Database Middleware, Informational Document. In: *Global Grid Forum (Frascati, Italien, 7.-10. Okt.)*, 2001

Newc02 Newcomer, E.: *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison Wesley, Boston u.a., 2002

ÖzVa99 Özsu, M.T; Valduriez, P.: *Principles of Distributed Database Dystems*. Prentice Hall, Englewood Cliffs, 1999

Rahm94 Rahm, E.: *Mehrrechner-Datenbanksysteme*. Addison Wesley, Bonn u.a., 1994

RiKT02 Risch, T.; Koparanova, M.; Thide, B.: High-Performance GRID Database Manager for Scientific Data. In: *Proceedings of the Workshop on Distributed Data & Structures (WDAS 2002, University Paris 9 Dauphine, Frankreich, 20.-23. März)*, 2002

ScLe02 Schlesinger, L.: Lehner, W.: Extending Data Warehouses by Semi-Consistent Database Views. In: *Proceedings of the 4th International Workshop on Design an Management of Data Warehouses (DMDW'02, Toronto, Canada, 27. Mai)*, 2002, S. 43-51

SnAh85 Snodgrass, R.T.; Ahn, I.: A Taxonomy of Time in Databases. In: *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data (SIGMOD'85, Austin, Texas, 28.-31. Mai)*, 1985, S. 236-246

TCG+93 Tansel, A. U.; Clifford, J.; Gadia, S.; Jajodia, S.; Segev, A.; Snodgrass, R.: *Temporal Databases: Theory, Design and Implementation*. The Benjamin/Cummings Publishing Company Inc., Redwood City (CA) u.a., 1993

WEBFLOW: Ein System zur flexiblen Ausführung webbasierter, kooperativer Workflows

Ulrike Greiner Erhard Rahm

Institut für Informatik, Universität Leipzig
http://dbs.uni-leipzig.de

Abstract: Der zunehmende Einsatz von Web-Technologien steigert die Notwendigkeit, kooperative Geschäftsprozesse und Workflows zwischen verschiedenen Unternehmen oder Abteilungen webbasiert abzuwickeln. Eine treibende Kraft hierfür ist die wachsende Verfügbarkeit von Web-Service-Realisierungen zur Interoperabilität. Die derzeit verfügbaren Ansätze unterstützen jedoch keine ausreichende Qualität und Flexibilität kooperativer webbasierter Workflows, insbesondere bzgl. der Ausführungsqualität von Diensten und der dynamischen Ausnahmebehandlung. Diese Defizite sollen mit dem System WEBFLOW abgebaut werden. Es ermöglicht die Definition und Überwachung von Ausführungsbedingungen für unterschiedlich mächtige Dienste verschiedener Kooperationspartner. Zudem unterstützt es eine regelbasierte dynamische Ausnahmebehandlung, mit der die Robustheit kooperativer Workflows deutlich verbessert werden soll.

1 Einleitung

Workflow-Management-Systeme unterstützen die Definition und Ausführung sich oft wiederholender und in ihrer Struktur weitgehend gleichbleibender Geschäftsprozesse und haben in Unternehmen, Kliniken oder Verwaltungen weite Verbreitung gefunden [GHS95], [JB96], [La97]. Diese Geschäftsprozesse und Workflows sind zunehmend kooperativ zwischen Unternehmen oder verschiedenen, auch geografisch verteilten Abteilungen abzuwickeln, die typischerweise unterschiedlichste Workflow-Systeme, Anwendungssysteme und Datenbanken einsetzen. Durch die weitgehende Web-Anbindung von Unternehmen und Einrichtungen sowie die zunehmende Verfügbarkeit von Web-Service-Implementierungen besteht nun erstmals die Möglichkeit, in größerem Umfang *kooperative Workflows* zu realisieren [LRS02]. Darunter verstehen wir Workflows, deren Aktivitäten durch Dienste unterschiedlicher Organisationseinheiten ausgeführt werden, wie die Bearbeitung eines Bauantrags, die verteilte Konstruktion von Fahrzeugen oder die Behandlung eines Patienten durch verschiedene niedergelassene Ärzte und Kliniken.

Derzeitige Web-Service-Realisierungen und -Standards decken primär die Sicherstellung einer Basis-Interoperabilität auf Ebene der Nachrichtenprotokolle und Austauschformate und zugehörige Aufgaben wie Registrierung von Diensten ab. Web Services werden derzeit auch primär zur Realisierung einfacher Dienste mit relativ kurzer Ausführungszeit verwendet, z.B. zur Ausführung einer einfachen Anwendungsfunktion oder einer Datenbankabfrage. Kooperative Workflows erfordern jedoch die Zusammenarbeit zwischen

komplexeren Diensten, die selbst durch einen Workflow realisiert sein können und durch oft lange Ausführungsdauer, Nutzerinteraktionen etc. gekennzeichnet sind. Besondere Herausforderungen bei der Realisierung der Workflow-Kooperation entstehen u.a. durch die Komplexität der einzubindenden Dienste, die Autonomie der beteiligten Unternehmen z.B. bzgl. der Implementierung ihrer Dienste sowie durch die für eine breite Nutzerakzeptanz notwendige Robustheit kooperativer Workflows. Deshalb ist beispielsweise eine ausreichende Flexibilität, insbesondere zur Behandlung von Fehlern und Ausnahmen, von größter Bedeutung. Neben erweiterten Recovery-Verfahren mit der Möglichkeit kompensationsbasierter Zurücksetzung von Teilschritten aufgrund von Systemausfällen und sonstigen technischen Fehlern werden flexible Mechanismen zur Behandlung logischer Ausnahmesituationen (Produktangebot überschreitet Preislimit, Liefertermin wird überschritten, etc.) benötigt. Bei diesen soll ein betroffener kooperativer Workflow nach der notwendigen Ausnahmebehandlung weiter „nach vorne“ ausgeführt werden können. Weiterhin sollten möglichst viele der Ausnahmen automatisch erkannt und behandelt werden können, um auch bei sehr häufiger Ausführung kooperativer Workflows eine hohe Robustheit und Ausführungsqualität zu erreichen.

Zur Umsetzung dieser Ziele entwickeln wir derzeit das webbasierte System WEBFLOW, dessen Grobentwurf in diesem Beitrag vorgestellt wird. WEBFLOW basiert auf folgenden Eigenschaften:

- Definition und Ausführung kooperativer Workflows im Rahmen einer Mediatorarchitektur, bei der heterogene Dienste über Organisationsgrenzen hinweg eingebunden werden und gleichzeitig die Autonomie der Kooperationspartner gewahrt bleibt;
- weitgehende Nutzung vorhandener Web-Service-Standards wie WSDL (Web Services Description Language, [Ch01]), UDDI (Universal Description, Discovery and Integration [Be02]) oder BPEL4WS (Business Process Execution Language for Web Services [Cu02]) und verfügbarer Implementierungen zur Gewährleistung der Interoperabilität und Reduzierung des Implementierungsaufwandes;
- Kooperationsmodell mit expliziter Spezifikation und Überwachung von temporalen und inhaltlichen Ausführungsbedingungen durch entsprechende Erweiterung von Diensten (Web Services). Die erweiterte Funktionalität wird durch den WEBFLOW-Mediator erbracht und kann sowohl für einfache Dienste (z.B. auf Basis von Legacy-Anwendungen) als auch für komplexe Dienste genutzt werden.
- Regelbasierte Erkennung und (teil-) automatische Behandlung von Ausnahmen mit flexiblen Reaktionsmöglichkeiten, insbesondere vorwärts orientierter Fortsetzung der Verarbeitung und dynamischer Adaption betroffener Workflows. Die Ausnahmebehandlung für einen Dienst kann für alle Workflows zentral im WEBFLOW-Mediator oder spezifisch in dem den Dienst verwendenden (kooperativen) Workflow festgelegt werden.

Im Folgenden werden nach einem Überblick über verwandte Arbeiten (Abschnitt 2) die Mediatorarchitektur in Abschnitt 3 und die Ausnahmebehandlung in Abschnitt 4 vorgestellt. Den Schluss bilden Zusammenfassung und Ausblick (Abschnitt 5).

2 Verwandte Arbeiten

In letzter Zeit beschäftigen sich verschiedene Forschungsarbeiten mit interorganisatorischen Workflows und der verteilten Ausführung von Workflows [AW01],[BRH02], [CS01],[Gr01],[La01],[Mu98],[SS01],[VW99]. Diese Ansätze bieten jedoch meist keine ausreichenden Möglichkeiten zur automatischen Überwachung von Ausführungsbedingungen bzw. unterstützen keine heterogenen Kooperationspartner mit unterschiedlich mächtigen Diensten. Eine Ausnahmebehandlung ist wenn überhaupt nur in einfacher Form vorhanden, typischerweise für die Überwachung von Deadlines. Einige in früheren Forschungsarbeiten vorgeschlagenen Konzepte zur dynamischen Ausnahmebehandlung sollen für die spezifischen Zielsetzungen von WEBFLOW angepasst werden, insbesondere die Verwendung einer regelbasierten Ausnahmeerkennung sowie dynamische Workflow-Adaptionen [MR00], [Re01].
Die Workflow Management Coalition hat mit Interface 4 eine Schnittstelle für die Kooperation verschiedener Workflow-Engines spezifiziert [Wf00], die sich aber auf den Datenaustausch konzentriert und keine Ausnahmebehandlungsmöglichkeiten enthält.
Web Services auf Basis von WSDL [Ch01] ermöglichen nur eine einfache Fehlerbehandlung durch vordefinierte Fehlernachrichten, unterstützen jedoch keine Ausführungsbedingungen. Die kürzlich erschienenen Standardisierungsvorschläge *BPEL4WS* (*Business Process Execution Language for Web Services* [Cu02]) und *WS-Transaction* [Ca02] zur Definition von Geschäftsprozessen auf der Basis von Web Services sehen eine Transaktionsunterstützung und eine kompensationsbasierte Ausnahmebehandlung beim Empfang vordefinierter Fehlernachrichten oder der Verletzung von Timeouts vor; ähnliches gilt für das Workflow-System Microsoft BizTalk Server [Me01], [Ro01]. Diese Ansätze können nur auf Ausnahmen reagieren, für die vordefinierte Fehlernachrichten existieren; zudem kann die Ausnahmebehandlung für einen Dienst nicht zentral festgelegt werden, sondern muss in jedem den Dienst verwendenden Workflow spezifiziert werden.

3 WEBFLOW-Architektur und Metadatenmodell

Wir beschreiben zunächst die Grobarchitektur von WEBFLOW und skizzieren danach die verwendeten Metadaten und das Kooperationsmodell.

3.1 Architekturüberblick

WEBFLOW basiert auf einer Mediatorarchitektur (Abb. 1), die analog zu Mediatoransätzen für die Integration heterogener Datenquellen die Integration heterogener Dienste verschiedener Organisationseinheiten in kooperative Workflows unter Wahrung der Autonomie der Anbieter unterstützt. Die zentrale Komponente, der WEBFLOW-Mediator, verfügt über die Funktionalität eines Workflow-Systems zur Ausführung kooperativer Workflows, in deren Rahmen unterschiedliche Dienste derselben Organisationseinheit oder externer Knoten eingebunden werden. Für diese Dienste können zur Verbesserung der Kooperationsqualität Ausführungsbedingungen definiert und überwacht werden; zur Verbesserung der Ausführungsflexibilität erfolgt eine (teil-) automatische Ausnahmebehandlung.
Die Ausführung der Dienste erfolgt typischerweise durch Anwendungssysteme und Da-

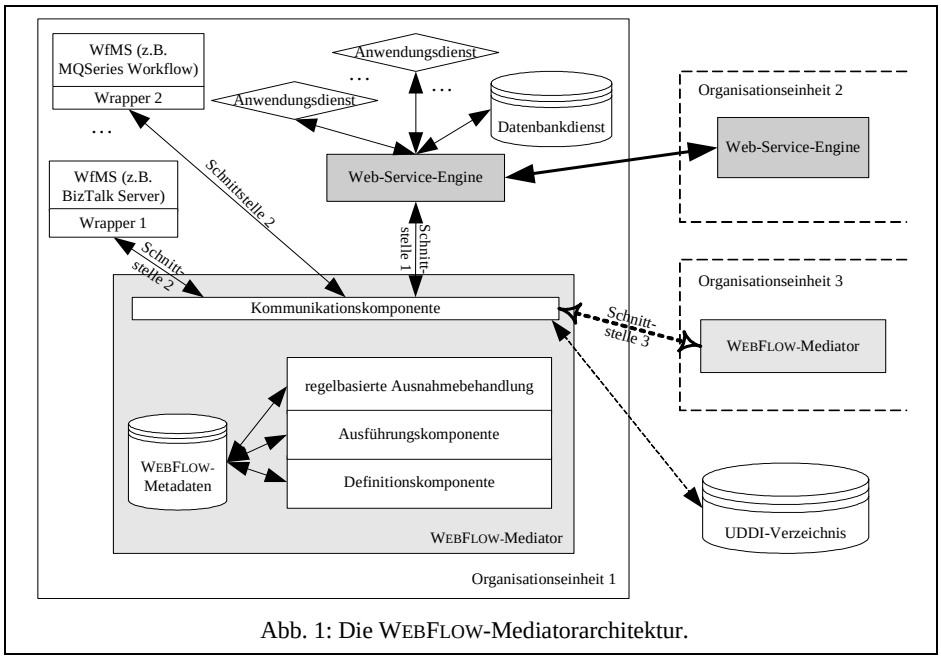


Abb. 1: Die WEBFLOW-Mediatorarchitektur.

tenbanksysteme, die vom Mediator über eine Web-Service-Engine angesprochen werden (Schnittstelle 1). Da Workflows derzeit meist nicht als Web Services definiert werden können, unterstützt WEBFLOW eine erweiterte WSDL-ähnliche Dienst-Schnittstelle für Workflow-Management-Systeme (WfMS) (Schnittstelle 2).

Wie Abb. 1 zeigt, besteht der WEBFLOW-Mediator aus fünf Hauptkomponenten:

- *Definitionskomponente* zur Definition von Diensten, Ausführungsbedingungen, Ausnahmeregeln und kooperativen Workflows. Einfache Dienste können dabei typischerweise aus UDDI-Verzeichnissen, komplexe Dienste von Mediatoren externer Kooperationspartner importiert werden.
- *Ausführungskomponente* zur Ausführung der kooperativen Workflows einer Organisationseinheit und zur Überwachung von Ausführungsbedingungen während der Workflow-Ausführung,
- *regelbasierte Ausnahmebehandlung*, die mit Hilfe spezieller ECA-Regeln die Ausnahmeerkennung und -behandlung durchführt,
- den *WEBFLOW-Metadaten* mit Informationen zu kooperativen Workflows, Diensten, Ausführungsbedingungen und Ausnahmen (s. 3.2),
- *Kommunikationskomponente*, die für den Mediator die unterschiedlichen Schnittstellen zu anderen Systemen verbirgt.

Kooperative Workflows werden ausgeführt, indem die Ausführungskomponente entsprechend des Kontrollflusses den oder die nächsten auszuführenden Dienste bestimmt und aufruft. Der Dienstaufruf kann sowohl synchron als auch asynchron erfolgen. Bei lokalen Diensten erfolgt der Aufruf über die lokale Web-Service-Engine (Schnittstelle 1) oder geht direkt an ein WfMS (Schnittstelle 2). Der Aufruf externer Dienste erfolgt entweder über die lokale Web-Service-Engine an die Web-Service-Engine des Partners oder direkt an

den WEBFLOW-Mediator des Partners (Schnittstelle 3). Über die Schnittstelle 3 werden unter anderem auch Metadaten zu Diensten externer Organisationseinheiten oder Nachrichten bei der Verletzung von Ausführungsbedingungen, die der Mediator des Partners überwacht, ausgetauscht (s. Abschnitt 4.1 zur Überwachung von Ausführungsbedingungen bei der Ausführung von Diensten).
Um eine große Flexibilität bzgl. der Ausnahmebehandlung zu erreichen, ist geplant als Workflow-Engine für WEBFLOW ADEPT$_{flex}$ [Re01] einzusetzen, das Operationen zur manuellen, dynamischen Workflow-Adaption anbietet. Außerdem werden in WEBFLOW Teile eines im Rahmen eines DFG-Projekts entwickelten Prototypen zur ereignisbasierten, dynamischen Workflow-Adaption verwendet (u.a. für die regelbasierte Ausnahmeerkennung).

3.2 WEBFLOW-Metadaten

Eine wichtige Komponente der WEBFLOW-Architektur sind die WEBFLOW-Metadaten, die unter anderem Informationen über kooperative Workflows, beteiligte Dienste, Ausführungsbedingungen, Ausnahmen und Regeln zur Ausnahmebehandlung enthalten. Sie werden z.B. bei der Überwachung von Ausführungsbedingungen oder für die regelbasierte Ausnahmebehandlung verwendet (Abschnitt 4). Abbildung 2 zeigt einen Ausschnitt des Metadaten-Modells in UML-Notation.
Die WEBFLOW-Metadaten umfassen alle notwendigen Informationen zu *kooperativen Workflows*, die aus *Diensten* bestehen, die von unterschiedlichen *Organisationseinheiten* ausgeführt werden. Dienste werden in *nicht-interaktive Dienste* (üblicherweise Web Services, die nach Aufruf ohne Nutzerinteraktion ausgeführt werden) und *interaktive Dienste*

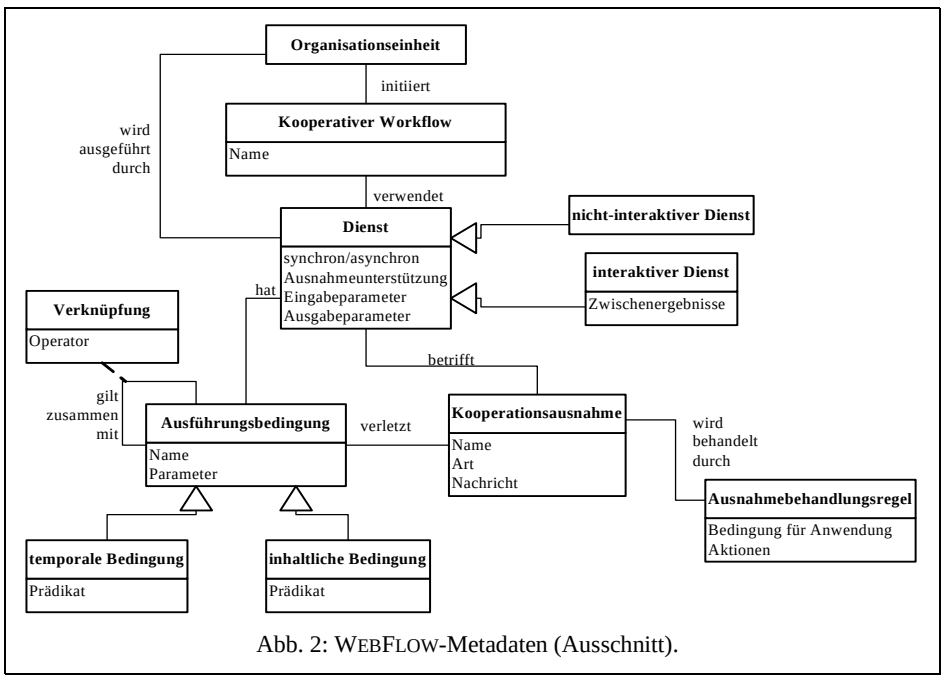


Abb. 2: WEBFLOW-Metadaten (Ausschnitt).

(typischerweise Workflows, die auch Zwischenergebnisse zurückgeben) unterteilt. Alle Arten von Diensten werden durch ihre *Aufrufart* (synchron/asynchron), ihre *Ein-* und *Ausgabeparameter* und den Grad der *Ausnahmeunterstützung* (siehe 3.3) charakterisiert.
Zur Qualitätskontrolle der Workflow-Kooperation können zu Diensten *Ausführungsbedingungen* definiert werden. Wir unterscheiden *temporale* Bedingungen wie Zeitpunkte („Auftrag muss bis 02.03.2003 bearbeitet werden“) oder Intervalle („Ergebnis muss innerhalb von 5 Tagen vorliegen“) und *inhaltliche* Bedingungen („Preis < 100 EUR“) oder Vorgaben („Gutachten muss Aussage zu xy enthalten“). Ausführungsbedingungen beziehen sich auf Ein- oder Ausgabeparameter eines Dienstes (Attribut *Parameter*) und spezifizieren einen zulässigen Wertebereich. Hat ein Dienst mehrere Ausführungsbedingungen, so werden diese mit booleschen Operatoren verknüpft (Assoziation *gilt zusammen mit*, Klasse *Verknüpfung*). Die Abbildung der Ausführungsbedingungen in einer eigenen Klasse erleichtert ihre Überwachung durch die erweiterte Ausführungskomponente des WEBFLOW-Mediators. Außerdem entsteht durch die Ausführungsbedingungen eine Klassifikation der Dienste, die z.B. die Suche nach bestimmten Diensten bei der Definition kooperativer Workflows erleichtert (z.B. „Finde einen Dienst, der den Auftrag innerhalb drei Tagen bearbeiten kann“).
Die für die Ausführungsflexibilität wesentliche Ausnahmebehandlung erfordert die automatische Erkennung und Behandlung von *Kooperationsausnahmen*. Die Kooperationsausnahmen bilden den Event-Teil der ECA-Regeln für die regelbasierte Ausnahmebehandlung. Condition- und Action-Teil stehen in der Klasse *Ausnahmebehandlungsregel*, die eine *Bedingung* für die Anwendung der Regel sowie die notwendigen *Aktionen* enthält. Abbildung 3 zeigt eine solche ECA-Regel für die Ausnahme, dass ein Fertigstellungstermin für ein Gutachten nicht eingehalten werden kann (Ereignis) und das Gutachten dringend benötigt wird (Bedingung, zu den Aktionen s. Abschnitt 4.2).

EREIGNIS	Terminüberschreitung durch Sachverständigen
BEDINGUNG	hohe Dringlichkeit des Auftrags
AKTIONEN	Suche alternativen Sachverständigen. Falls ein solcher existiert: Abbruchnachricht an verspäteten Dienst. Passe Fristen in den betroffenen kooperativen Workflows an.

Abb. 3: Beispielregel zur Ausnahmebehandlung.

Zu einer Kooperationsausnahme kann es mehrere ECA-Regeln geben, um die Ausnahmebehandlung in Abhängigkeit von konkreten Laufzeitdaten zu spezifizieren. Durch die Definition der Ausnahmebehandlung auf der Ebene der Metadaten müssen die Regeln für jeden Dienst nur einmal definiert werden und nicht redundant in allen kooperativen Workflows enthalten sein, die diesen verwenden. Damit kann insbesondere eine vorwärts orientierte Ausnahmebehandlung unterstützt werden, außerdem wird die Übersichtlichkeit der kooperativen Workflows erhöht.

3.3 Ausnahmeunterstützung

Wie bereits im vorigen Abschnitt dargestellt werden Dienste, die in einem kooperativen

Workflow verwendet werden, u.a. durch den Grad der sogenannten *Ausnahmeunterstützung* charakterisiert. Diese beschreibt welche Informationen über den Aufruf und die Ein- und Ausgabeparameter hinaus zwischen Diensten und dem WEBFLOW-Mediator fließen und für die Ausnahmeerkennung genutzt werden können. Wir unterscheiden drei Grade:

- Grad 0 - keine Ausnahmeunterstützung:
  keine Fehlernachrichten, keine Ausführungsbedingungen
- Grad 1 - Standard-Ausnahmeunterstützung:
  Fehlernachrichten bei erfolgloser Ausführung, keine Ausführungsbedingungen
- Grad 2 - erweiterte Ausnahmeunterstützung:
  Fehlernachrichten und temporale oder inhaltliche Ausführungsbedingungen

Grad 0 gilt für Dienste, die keinerlei Fehlermeldungen schicken, z.B. einfache Applikationen, die in Web Services gekapselt wurden. Grad 1 beschreibt Dienste, die vordefinierte Fehlernachrichten bei Ausführungsfehlern verschicken (z.B. „unvollständigen Daten für die Bearbeitung einer Anfrage“, „Artikel vergriffen“). Solche Fehlernachrichten sind z.B. die fault-Nachrichten einer WSDL-Web-Service-Definition. Unterstützt ein Dienst zusätzlich temporale oder inhaltliche Ausführungsbedingungen entspricht er Grad 2.
Die obige Klassifikation ist orthogonal zur Einteilung in nicht-interaktive und interaktive Dienste in Abschnitt 3.2; derzeit sind jedoch nicht-interaktive Dienste, die über eine Standard-Web-Service-Schnittstelle aufgerufen werden, typischerweise vom Grad 0 oder 1. Grad 2 hingegen ist überwiegend bei interaktiven Diensten von Bedeutung, da hier eine erweiterte Schnittstelle und zusätzliche Funktionalität beim Anbieter zur Überwachung der Ausführungsbedingungen erforderlich sind. Dadurch können Ereignisse, die voraussichtlich zu einer Verletzung von Ausführungsbedingungen führen (wie die Erkrankung eines Sachverständigen), an den WEBFLOW-Mediator gemeldet werden *bevor* die eigentliche Verletzung eintritt. Zur Erhöhung der Kooperationsqualität sind also Dienste vom Grad 2 wünschenswert. Da aber derzeit nur wenige Dienste diesen Grad unterstützen, bietet WEBFLOW die Möglichkeit, Ausführungsbedingungen auch für Dienste mit Ausnahmeunterstützung vom Grad 0 oder 1 zu definieren und zu überwachen ohne in die Anwendungen, die die Dienste ausführen, einzugreifen, was insbesondere bei Diensten externer Organisationseinheiten von Vorteil ist.

4 Dynamische Ausnahmebehandlung

Die dynamische Ausnahmebehandlung von WEBFLOW gliedert sich in die Erkennung einer Kooperationsausnahme und die Reaktion auf die Ausnahme. Ziel ist es, die Ausführung eines kooperativen Workflows auch nach Kooperationsausnahmen möglichst „nach vorne“ fortzusetzen. Außerdem sollen von der Ausnahme betroffene Dienste frühzeitig über zu erwartende Verzögerungen oder Probleme informiert werden. Die zuverlässige Ausführung verteilter Workflows auf Basis transaktionaler Dienste und verteilter Commit-Protokolle ist orthogonal zur dynamischen Ausnahmebehandlung und wird hier nicht weiter betrachtet; hierzu sollen an *WS-Transaction* [Ca02] angelehnte Realisierungen verwendet werden.

4.1 Ausnahmeerkennung

Welche Schritte zur *Erkennung* einer Ausnahme nötig sind, wird von der Art der Ausnahme und vom Grad der Ausnahmeunterstützung eines Dienstes (vgl. 3.3) bestimmt. Ausnahmen entstehen entweder durch Ausführungsfehler eines Dienstes, durch die Verletzung von Ausführungsbedingungen oder durch andere, unerwartete Ereignisse, die die erfolgreiche Dienstausführung beeinflussen (wie z.B. Krankheit eines Sachverständigen). Tabelle 1 gibt einen Überblick, welche Möglichkeiten bei unterschiedlicher Ausnahmeunterstützung bestehen Ausführungsfehler und Bedingungsverletzungen zu erkennen.
Bei Diensten mit Ausnahmeunterstützung vom Grad 0, also ohne Fehlernachrichten oder Ausführungsbedingungen, können Ausnahmen aufgrund von Ausführungsfehlern (z.B. unkorrekte Ausführung, Absturz der Anwendung, Netzfehler, Überlastung) entweder daran erkannt werden, dass ein aufgerufener Dienst nicht innerhalb einer vorgegebenen Zeit antwortet oder Rückgabeparameter sinnlose Werte enthalten. Die Erkennung erfolgt also durch beim Dienstaufruf gesetzte Timeouts oder eine semantische Prüfung der Rückgabeparameter. Bei Standard- und erweiterter Ausnahmeunterstützung werden Ausführungsfehler durch vordefinierte Fehlernachrichten angezeigt (z.B. „Artikel nicht verfügbar“).
Die Verletzung von Ausführungsbedingungen wird bei Diensten vom Grad 0 und 1 durch die Überwachung der Bedingungen durch den Mediator erkannt. Temporale Bedingungen werden dabei mit Hilfe von Timeouts oder Deadlines und inhaltliche Bedingungen durch die Überprüfung der zurückgegebenen Ausgabeparameter überwacht (z.B. „Ist der Preis größer als das vorgegebene Limit?“). Damit werden Verletzungen erst erkannt, wenn der gesetzte Timeout verstrichen ist oder die Dienstausführung beendet und die Parameter zurückgegeben wurden. Dienste vom Grad 2, die ihre Ausführungsbedingungen selbst überwachen, schicken bei einer Verletzung Fehlernachrichten an den Mediator.
Ausnahmen der dritten Art, unerwartete Ereignisse, die die erfolgreiche Dienstausführung behindern, können nur bei interaktiven Diensten erkannt werden, da sie durch einen Nutzer z.B. in Form einer E-Mail oder eines Telefonanrufs gemeldet werden müssen; anschließend werden sie über eine Eingabemaske an den WEBFLOW-Mediator weitergeleitet.

4.2 Reaktion

WEBFLOW verwendet zur Ausnahmebehandlung eine mehrstufige Vorgehensweise. Wurde eine Kooperationsausnahme erkannt, werden zunächst anhand der in den Metadaten vorhandenen Informationen automatisch alle potenziell von einer Ausnahme *betroffenen kooperativen Workflow-Instanzen* bestimmt. Im Beispiel aus Abb. 3 könnten das alle Instanzen sein, die Gutachten hoher Dringlichkeit bei dem gleichen Sachverständigen in Auftrag gegeben haben und deren Fertigstellungstermine (die z.B. in einer Ausführungs-

	Ausführungsfehler	Bedingungsverletzung
Grad 0 - keine	Timeouts / semantische Prüfung	Überwachung von Ausführungsbedingungen
Grad 1 - Standard	vordefinierte Fehlernachrichten	
Grad 2 - erweitert	vordefinierte Fehlernachrichten	Fehlernachrichten

Tab. 1: Erkennung von Ausnahmen in Abhängigkeit von der Ausnahmeunterstützung.

bedingung abgelegt sind) innerhalb der nächsten 24 Stunden liegen.
Danach wird in den Metadaten eine passende *Regel* mit den notwendigen Aktionen gesucht. Wird keine passende Regel gefunden, wird geprüft, ob in der betroffenen Workflow-Instanz eine Aktion definiert ist und gegebenenfalls zur Anwendung gebracht. Wurde auch dort nichts spezifiziert, erfolgt die Benachrichtigung eines verantwortlichen Nutzer, so dass dieser manuell auf die Ausnahme reagieren kann.

Die möglichen Reaktionen auf eine Ausnahme, die im Aktionsteil der Regeln festgelegt sind, hängen stark vom verwendeten Workflow-Modell und den Fähigkeiten der verwendeten Workflow-Engine ab:

- Bei den meisten kommerziellen Workflow-Systemen kann man den Workflow nach Auftreten einer Ausnahme nur abbrechen und/oder eine manuelle Ausnahmebehandlung durchführen.
- Manche Systeme (z.B. Microsofts BizTalk Server) bieten die Möglichkeit Kompensationsprozesse zu definieren, die beispielsweise einen Dienst erneut aufrufen oder einen alternativen Dienst ausführen, wenn eine Dienstausführung fehlschlägt.
- Ein weiterer Schritt zur Erhöhung der Flexibilität ist eine manuelle oder automatische Workflow-Adaption, bei der z.B. Aktivitäten gelöscht oder eingefügt oder Zeitkanten, die den zeitlichen Abstand zwischen zwei Aktivitäten festlegen, bei Verzögerungen angepasst werden (vgl. [MR00]). Um solche Änderungen zur Laufzeit durchzuführen, benötigt man eine Workflow-Engine wie $\text{ADEPT}_{\text{flex}}$ [Re01], die dynamische Änderungen unterstützt.

WEBFLOW verwendet ein Workflow-Modell, mit dem z.B. Zeitabhängigkeiten modelliert werden können und eine Workflow-Engine, die dynamische Änderungen unterstützt. Die möglichen Reaktionen auf die in Abb. 3 dargestellte Ausnahme umfassen demzufolge die Suche nach einem alternativen Sachverständigen, eine Abbruchnachricht an den verspäteten Sachverständigen, falls ein alternativer Dienst gefunden wurde, sowie die Anpassung von Fristen in den betroffenen kooperativen Workflows. Mit Hilfe der ECA-Regeln und der übrigen Metadaten können die notwendigen Aktionen durch die Ausnahmebehandlungskomponente des WEBFLOW-Mediators automatisch bestimmt und ausgeführt werden. Die Ausführung kann zur Sicherheit von der Bestätigung eines verantwortlichen Nutzers abhängig gemacht werden.

5 Zusammenfassung und Ausblick

Im vorliegenden Beitrag wurden Grobarchitektur, Kooperationsmodell und Ausnahmebehandlung des Systems WEBFLOW überblicksartig vorgestellt. Wesentliche Zielsetzung ist auf Basis von Web Services die Qualität, Flexibilität und Robustheit webbasierter kooperativer Workflows zu erhöhen, insbesondere durch die Definition und Überwachung von Ausführungsbedingungen für unterschiedlich mächtige Dienste und durch eine regelbasierte, teilautomatische und vorwärts orientierte Ausnahmebehandlung.
Zur Umsetzung der vorgestellten Konzeption wird derzeit die Definitionssprache für kooperative Workflows mit Ausführungsbedingungen näher spezifiziert, die sich an dem Workflow-Modell von $\text{ADEPT}_{\text{flex}}$ und [MR00] sowie an BPEL4WS orientiert. Zur WEBFLOW-Realisierung wird wie schon erwähnt auf einen im Rahmen eines DFG-Projekts ent-

wickelten Prototypen zur ereignisbasierten, dynamischen Workflow-Adaption zurückgegriffen, der unter anderem um die Komponenten zur Überwachung der Ausführungsbedingungen sowie zur Web-Service-Anbindung erweitert wird.

Danksagung: Wir bedanken uns für die wertvollen Hinweise der anonymen Gutachter. Die Arbeit wurde durch die Deutsche Forschungsgemeinschaft (DFG) unterstützt, Projektnummer Ra497-12.

Literaturverzeichnis

[AW01] van der Aalst, W.M.P., Weske, M.: The P2P Approach to Interorganizational Workflows. In Proc. of CAiSE 2001, S. 140-156.

[Be02] Bellwood, T. et al.: UDDI Version 3.0, Juli 2002. http://uddi.org/pubs/uddi_v3.htm

[BRH02] Bon, M.; Ritter, N.; Härder, T.: Sharing Product Data among Heterogenous Workflow Environments. In Proc. of CAD 2002, Dresden, 2002, S.139-150.

[Ca02] Cabrera, F. et al.: Web Services Transaction (WS-Transaction), August 2002. http://www-106.ibm.com/developerworks/library/ws-transpec/

[CS01] Casati, F.; Shan, M.-C.: Dynamic and adaptive composition of e-services. Information Systems 26, 2001, S. 143-163.

[Ch01] Christensen, E. et al.: Web Services Description Language (WSDL) 1.1, W3C Note, March 2001. http://www.w3.org/TR/wsdl

[Cu02] Curbera, F. et al.: Business Process Execution Language for Web Services, Version 1.0 Juli 2002. http://www-106.ibm.com/developerworks/library/ws-bpel/

[GHS95] Georgakopoulos, D.; Hornick, M.; Sheth, A.: An Overview of Workflow Management: From Process Modeling to Infrastructure for Automation. Journal on Distributed and Parallel Database Systems 3(2), 1995; S. 119-153.

[Gr01] Grefen, P.; Aberer, K.; Ludwig, H.; Hoffner, Y.: CrossFlow: Cross-Organiziational Workflow Management for Service Outsourcing in Dynamic Virtual Enterprises. In IEEE Bull. of the Tech. Comm. on Data Engineering, 24(1), March 2001, S. 52-57.

[JB96] Jablonski, S.; Bussler, C.: Workflow Management. Modeling Concepts, Architecture, and Implementation. International Thomson Computer Press, London, 1996.

[La97] Lawrence, P.: Workflow Handbook. John Wiley and Sons, New York, 1997.

[La01] Lazcano, A. et al.: WISE: Process based E-Commerce. In IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 24, No.1, March 2001, S. 46-51.

[LRS02] Leymann, F.; Roller, D.; Schmidt M.-T.: Web services and business process management. In IBM Systems Journal, Vol. 41, No. 2, 2002, S. 198-211.

[Me01] Mehta, B. et al.: BizTalk Server 2000 Business Process Orchestration. In IEEE Bulletin of the Tech. Committee on Data Engineering, Vol. 24, No. 1, March 2001, S. 35-39.

[MR00] Müller, R.; Rahm, E.: Dealing with Logical Failures for Collaborating Workflows. In Etzion, O.; Scheuermann, P. (Hrsg.): Proc. of CoopIS 2000, Eilat, 2000. LNCS 1901, Springer: S. 210-233.

[Mu98] Muth, P. et al.: Enterprise-wide workflow management based on state and activity charts. In Dogac, A. et al. (Hrsg.): Workflow management systems and interoperability. NATO Advanced Study Institute, Springer, New York, 1998, S. 281-303.

[Re01] Reichert, M. et al.: Realisierung flexibler, unternehmensweiter Workflow-Anwendungen mit ADEPT. In P. Horster (Hrsg.): Proc. Elektr. Geschäftsprozesse - Grundlagen, Sicherheitsaspekte, Realisierungen, Anwendungen. Klagenfurt, 2001, S. 217-228.

[Ro01] Roxburgh, U.: BizTalk Orchestration: Transactions, Exceptions and Debugging. Microsoft White Paper, 2001.

[SS01] Schönhoff, M.; Stormer, H.: Trading Workflows Electronically: The ANAISOFT Architecture. In Proc. der BTW 2001, Oldenburg, 2001, S. 67-74.

[VW99] Vossen, G.; Weske, M.: The WASA2 Object-Oriented Workflow Management System. In Proc. ACM SIGMOD Conference, Philadelphia, 1999, S. 587-589.

[Wf00] Workflow Management Coalition: Workflow Standard-Interoperability. Wf-XML Binding. Doc. Nr. WFMC-TC-1023, 2000. http://www.wfmc.org

Modellierung und Abwicklung von Datenflüssen in unternehmensübergreifenden Prozessen

Markus Bon
AG DBIS
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-67653 Kaiserslautern
bon@informatik.uni-kl.de

Norbert Ritter
Arbeitsbereich VSIS
Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Strasse 30
D-22527 Hamburg
ritter@informatik.uni-hamburg.de

Hans-Peter Steiert
Research &Technology
RIC/ED
DaimlerChrysler AG
Postfach 2360
D-89013 Ulm
hans-peter.steiert@daimlerchrysler.com

Zusammenfassung: Workflow-Management ist ein mittlerweile wohl erforschtes Gebiet, soweit es um die Möglichkeiten einzelner Workflow-Management-Systeme (WfMS) geht. Eine automatisierte Kontrolle von Abhängigkeiten zwischen von heterogenen WfMS gesteuerten Prozessen ist dagegen weitgehend unerforscht. Eine solche Kontrolle ist jedoch hinsichtlich der Steuerung von firmenübergreifenden Prozessen als äußerst wünschenswert zu betrachten, da eine beträchtliche Aufwandsreduzierung zu erwarten ist. Dieser Artikel leistet einen Beitrag zur Erschließung dieses Gebiets durch Beschreibung eines Ansatzes zur automatisierten Abwicklung von Datenflüssen zwischen heterogenen Workflows. Es werden die wesentlichen, für die Entwicklung einer entsprechenden Integrationskomponente relevanten Aspekte aufgezeigt und ein auf EAI-Technologie basierender Architekturansatz vorgestellt.

1 Einleitung

In großen Unternehmen existieren eine Vielzahl verschiedener, oftmals sehr komplexer Prozesse. Aufgrund der Forderungen nach immer kürzeren Entwicklungszeiten wird die Entwicklung nicht mehr allein im Unternehmen selbst durchgeführt, sondern es wird immer öfter auf das Fachwissen spezialisierter Firmen zugegriffen. Gerade zur Steuerung solcher komplexer, unternehmensübergreifender Prozesse bietet Workflow-Management-Technologie ein großes Nutzenpotential.

Im Projekt COW (Cross-Organizational-Workflows [KRS01]) wurde bereits untersucht, inwieweit sich inselübergreifende Prozesse beschreiben lassen, und wie ein globaler Kontrollfluss umgesetzt werden kann. Der Begriff der „Insel“ bezieht sich dabei auf die Gesamtheit der in einer Firma vorhandenen Systeme (Workflow-Management-Systeme, Produktdaten-Management-Systeme, etc.), Prozess- und Workflow-Typen sowie Ressourcen (Applikationen und/oder organisatorische Einheiten). Ein in diesem Zusammenhang äußerst wichtiger Aspekt ist der inselübergreifende Datenfluss [BHR01, BRH02].

Für eine sinnvolle Zusammenarbeit ist es notwendig, dass Daten zwischen den beteiligten Gruppen fließen. Beispielsweise müssen an einer Konstruktion beteiligte Ingenieure CAD-Geometrien austauschen, um die Passgenauigkeit der Einzelteile überprüfen zu können. Die automatisierte Abwicklung von solchen inselübergreifenden Datenflüssen ist unser Ziel.

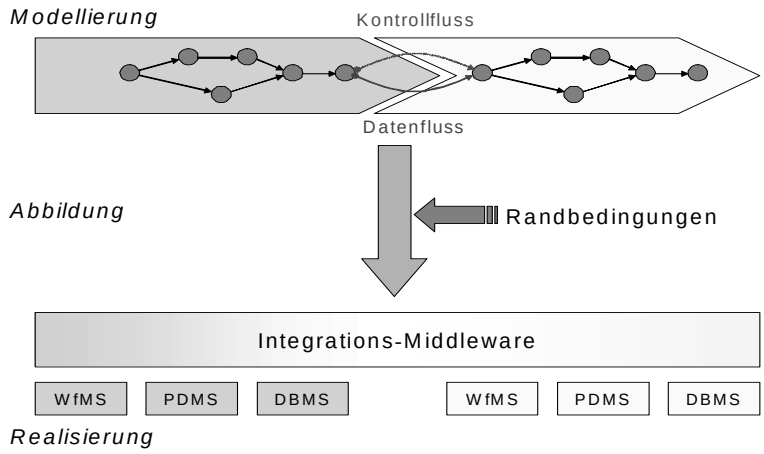


Abbildung 1: Modellierung und Realisierung von Datenflüssen

Um dieses Ziel zu erreichen, sind die folgenden Probleme zu lösen:

- Die Modellierung (siehe Abbildung 1, oberer Teil) unternehmensübergreifender Prozesse muss angemessen unterstützt werden, so dass Datenflussabhängigkeiten (DfA) zwischen den Inseln und die daran geknüpften Eigenschaften hinreichend beschrieben werden können. Hierzu sind Kategorien von Datenflüssen zu identifizieren.
- Für jede Kategorie ergeben sich verschiedene Realisierungsmöglichkeiten. Um zu einer gegebenen Systemlandschaft und organisatorischen Randbedingungen die passende Realisierung bestimmen zu können, ist es notwendig, Randbedingungen zu beschreiben und Implementierungsrichtlinien zu entwickeln (siehe Abbildung 1, mittlerer Teil).
- Letztlich wird eine Middleware benötigt, die eine integrierte Sicht auf unternehmensübergreifende Workflows erlaubt. In unserem Fall bedeutet dies, dass der Datenfluss im Rahmen von unternehmensübergreifenden Prozessen automatisiert abgewickelt wird (siehe Abbildung 1, unterer Teil). Dabei halten wir es für eine wichtige Anforderung, dass die Realisierung möglichst ohne größere Eingriffe in die beteiligten Infrastrukturen erfolgen soll.

In diesem Papier diskutieren wir die wesentlichen Aspekte des inselübergreifenden Datenflusses und leiten daraus einen Satz von sinnvollen Kategorien (Integrationsmuster) ab. Anschließend erläutern wir, dass zu einem Integrationsmuster unterschiedliche Möglichkeiten der Realisierung ins Auge gefasst werden müssen, da sich die Realisierung an den tatsächlichen Gegebenheiten der beteiligten Infrastrukturen orientieren muss. Weiterhin werden wir aufzeigen, wie sich heutige WfMS- und EAI-Werkzeuge zusammen mit ergänzenden Komponenten verwenden lassen, um die benötigte Middleware bereitzustellen, auf deren Basis sich die Datenflüsse realisieren lassen. Wir schließen das Papier mit einer Zusammenfassung und einem Ausblick.

2 Identifikation der Klassifikationskriterien

Um einen inselübergreifenden Fluss von Kooperationsdaten definieren zu können, benötigen wir einen Satz von Kriterien, mit denen sich die Eigenschaften des Datenflusses beschreiben lassen. Dazu sind im Einzelfall folgende Fragestellungen zu betrachten, die in den Abschnitten 2.1 bis 2.4 detailliert weiterverfolgt werden.

- In welcher Form werden die Daten vor dem eigentlichen Datenfluss verwaltet?
- Welche Wirkung hat die Durchführung des Datenflusses sowohl auf der Quell- als auch auf der Zielseite?
- Welche Auswirkungen hat der Datenfluss auf die Besitz- und Eigentumsverhältnisse?
- In welcher Form liegen die Daten vor und wie können sie der Zielseite bereitgestellt werden?

Wir gehen für unsere Betrachtungen zunächst von einem vereinfachten Szenario aus. Wir nehmen dabei an, dass lediglich zwei Insel-Systeme miteinander über eine DfA verknüpft werden sollen. Auf beiden Inseln befindet sich jeweils ein WfMS zur automatisierten Unterstützung lokaler Prozesse. Insbesondere bearbeiten diese Systeme auch diejenigen Workflows, die als Quell- bzw. Ziel-Workflows globaler DfAs zu betrachten sind. Weiterhin befinden sich auf beiden Inseln Applikationen, mit deren Hilfe prozessrelevante Tätigkeiten durchgeführt werden sollen. Die zu verarbeitenden Daten werden in der Regel persistent gespeichert. Daher befindet sich auf jeder der Inseln auch ein Datenhaltungssystem (DS_Q und DS_Z). Als weitere Komponente ist die übergeordnete Schicht zur Insel-übergreifenden Workflow-Integration (WfI) zu nennen. Diese repräsentiert alle Mechanismen, die benötigt werden, um definierte DfAs zu überwachen, aufzulösen und für das Bereitstellen der Kooperationsdaten in der gewünschten Form zu sorgen. Sie stellt dazu die Funktionalität zur Verfügung, die zusätzlich zu den auf den einzelnen Inseln bereits zur Verfügung stehenden Möglichkeiten benötigt wird.

Die Diskussionen in den nachfolgenden Abschnitten beziehen sich auf dieses Szenario und wir gehen davon aus, dass ein Prozessschritt, der durch $WfMS_Q$ angestoßen und durch A_Q bearbeitet wurde, Daten produziert, die aufgrund einer definierten DfA von der Insel 1 zur Insel 2 ’fließen’ müssen.

2.1 Verwaltung der Daten

Wir sprechen von *DS-verwalteten Daten*, wenn die Daten nach ihrer Erzeugung im lokalen Datenhaltungssystem DS_Q gespeichert werden. In diesem Fall wird die Durchführung des globalen Datenflusses von Insel 1 nach Insel 2 einen Zugriff auf DS_Q erfordern.

Sollte es sich bei den Kooperationsdaten nicht, wie im ersten angesprochenen Fall, um Applikationsdaten sondern um Workflow-relevante Daten handeln, die, wie es die meisten gängigen WfMS unterstützen, von der Applikation an das WfMS zur weiteren Verwaltung übergeben werden, so sprechen wir von *WfMS-verwalteten Daten.*

2.2 Wirkung des Datenflusses

Nachdem wir die Verwaltung der Daten auf der Quell-Insel betrachtet haben, wollen wir nun die Abwicklung globaler Datenflüsse näher untersuchen. Wir unterscheiden den Datenfluss anhand seiner Wirkung auf Insel 1 zwei Varianten:

- *quellerhaltend:* die Daten stehen nach der Datenübertragung immer noch auf der Quellseite bereit.
- *quellverbrauchend* : die Daten werden auf der Quellseite entfernt.

Betrachten wir nun die möglichen Wirkungen des Datenflusses als Kombinationen der Verwalter auf beiden Inseln und der Wirkung auf Insel 1, so ergeben sich für das Übertragen der Kooperationsdaten folgende Varianten.

1. *Replizieren:* quellerhaltend von DS_Q nach DS_Z
2. *Kopieren*: quellerhaltend von DS_Q nach $WfMS_Z$
3. *Abgeben:* quellverbrauchend von DS_Q nach DS_Z
4. *Übergeben:* quellverbrauchendes von DS_Q nach $WfMS_Z$
5. *Verteilen:* quellerhaltend von $WfMS_Q$ nach $WfMS_Z$
6. *Eintragen:* quellerhaltend von $WfMS_Q$ nach DS_Z
7. *Reisen:* quellverbrauchend von $WfMS_Q$ nach $WfMS_Z$
8. *Ablegen:* quellverbrauchend von $WfMS_Q$ nach DS_Z

2.3 Eigentümer und Besitzer

Ein weiteres Unterscheidungsmerkmal ist die Frage nach Eigentümer und Besitzer der Daten. *Besitzer* von Daten ist grundsätzlich jeder, der auf diese in irgendeiner Form zugreifen kann und darf. *Eigentümer* von Daten zu sein bedeutet, die Kontrolle darüber zu haben, was mit diesen Daten geschieht, bzw. welche Version der Daten als gültig anzusehen ist. Der Eigentümer ist auch für die Gültigkeit und Konsistenz der von ihm angebotenen Daten verantwortlich. Ein Weitergeben von Daten muss dabei nicht den Verlust des Eigentums bedeuten. Es kann durchaus die Situation entstehen, dass der Eigentümer seine Daten zeitweise überhaupt nicht in Besitz hat. In Tabelle 1 haben wir eine Übersicht über die unserer Meinung nach relevanten Fälle zusammengestellt. Der Buchstabe E markiert den Eigentümer, B markiert den Besitzer.

2.4 Bereitstellungsmodus

Für die Bereitstellung der Kooperationsdaten auf der Zielinsel kommen zwei verschiedene Bereitstellungsmodi in Frage. Der Modus *materialisiert* beschreibt den Fall, dass die Kooperationsdaten physisch auf der Zielseite zum Zugriff bereitgestellt werden. Natürlich ist dabei zu beachten, dass gerade im CAX-Bereich sehr große Datenmengen anfallen können, deren physische Übertragung sich sehr aufwändig gestalten kann. Daher macht die materialisierte Bereitstellung nur dann Sinn, wenn

sicher ist, dass die Daten auf der Zielseite in vollem Umfang benötigt und zugegriffen werden.

Der Modus *referenziert* hingegen sieht zunächst lediglich die Übergabe einer Referenz vor, so dass die Daten dann, wenn sie tatsächlich gebraucht werden, anhand dieser Referenz angefordert werden können. Hierbei besteht zusätzlich die Möglichkeit einer Art Parametrisierung, so dass exakt die Datenmenge angefordert und bereitgestellt werden kann, die tatsächlich benötigt wird. Ein solches Konzept ist offenbar in dem Fall sinnvoll, in dem die tatsächlich benötigten Daten in ihrem Umfang a priori nicht vollständig spezifiziert werden können.

	Vorher		Nachher		
	Insel 1	Insel 2	Insel 1	Insel 2	Beschreibung
1	*E, B*	-	*E, B*	*B*	Kopie (K)
2	*E, B*	-	*B*	*E, B*	Kopie, Abgabe des Eigentumsrechts (KAE)
3	*E, B*	-	*E, B*	*E, B*	Kopie, Gewähr des Eigentumsrechts (KGE)
4	*E, B*	-	*E*	*B*	Übergabe (Ü)
5	*E, B*	-	-	*E, B*	Übergabe, Abgabe des Eigentumsrechts (ÜAE)
6	*B*	*E*	-	*E, B*	Rückgabe (R)
7	*B*	*E*	*B*	*E, B*	Rückgabe, Beibehaltung der Kopie (RBK)

Tabelle 1: Übersicht über Eigentümer/Besitzer-Konstellationen

2.5 Modellierung unternehmensübergreifender Datenflüsse

Nachdem wir in den Abschnitten 2.1 bis 2.4 wesentliche Merkmale von Inselübergreifenden Datenflüssen analysiert haben, führen wir diese nun zusammen und beschreiben in diesem Abschnitt die sinnvoll zu bildenden Kombinationen. Daraus ergibt sich, welche Kategorien von Datenflüssen eine Middleware unterstützen muss, die zur Realisierung herangezogen werden soll.

Aus den bisher beschriebenen Elementen lassen sich eine Vielzahl verschiedener Arten von Datenflüssen herleiten. Betrachten wir das Zusammenspiel zwischen Verwaltung auf der Quellseite, Bereitstellungsmodus, Wirkung der Datenübertragung und der Besitz-/Eigentumsrechte, so sind nicht alle theoretisch bildbaren Kombinationen auch sinnvoll möglich. Wenn die Wirkung des Datenflusses im Transfer einer Kopie besteht, dann muss die Quellinsel auch Besitzer der Daten bleiben; Besteht die Wirkung des Datenflusses in der Übergabe der Daten, dann darf die Quellinsel nicht Besitzer der Daten bleiben. Eine referenzierte Übertragung von auf der Quellseite WfMS-verwalteten Daten ist nicht sinnvoll, da WfMS in der Regel nur eine lokale Verwendung von Workflow-relevanten Daten unterstützen. Tabelle 2 gibt eine Übersicht über mögliche Integrationsmuster, die sich als sinnvolle Kombination hinsichtlich der in den Abschnitten 2.1 bis 2.4 beschriebenen Aspekte ergeben.

Verwaltung Quelle	Modus	Wirkung	E/B-Konstellation
DS	materialisiert, referenziert	Replizieren, Kopieren	K, KAE, KGE, RBK
		Abgeben, Übergeben	Ü, ÜAE, R
WfMS	materalisiert	Verteilen, Eintragen	K, KAE, KGE, RBK
		Ablegen, Reisen	Ü, ÜAE, R

Tabelle 2: Integrationsmuster

3 Realisierungsaspekte

Eine Diskussion der wesentlichen Einflussfaktoren eines Insel-übergreifenden Datenflusses und ihrer Kombinationsmöglichkeiten, wie sie im vorangegangenen Kapitel vorgenommen wurde, ist natürlich notwendig, um eine Middleware zu entwickeln, die solche Datenflüsse automatisiert abarbeiten kann.

Bei der Abwicklung eines gewünschten Datenflusses sind jedoch auf den beteiligten Inseln vorherrschende technische und organisatorische Randbedingungen zu berücksichtigen. Diese bestimmen die zur Abwicklung des Datenflusses tatsächlich notwendigen Aktionen. Daraus folgt wiederum, dass eine geeignete Middleware in der Lage sein muss, diese Randbedingungen in flexibler Weise zu berücksichtigen.

Anbindung der WfI an $WfMS_1$

Aktuelle WfMS unterstützen lokale Datenflüsse [LR00]. Soll die lokale Umgebung jedoch verlassen werden, wird eine Schnittstelle nach außen benötigt. Hierzu gibt es im wesentlichen zwei Möglichkeiten: Die lokalen Workflow-Typen werden um Aktivitäten erweitert, die die Zusammenarbeit mit der WfI übernehmen, oder die WfI bekommt die Möglichkeit, die Workflow-Abarbeitung durch das $WfMS_1$ zu überwachen. Im ersten Fall werden in die lokalen Workflow-Typen an den Stellen spezielle, neue Aktivitäten eingefügt, an denen Datenflüsse zu anderen Inseln abgewickelt werden müssen. Diese Aktivitäten rufen die WfI auf, übergeben die Referenz auf die Daten und lösen so den Datenfluss aus. Im zweiten Fall überwacht die WfI beispielsweise den Zustand relevanter Workflow-Instanzen. Wird das Ende einer Aktivität erkannt, deren Ergebnis übertragen werden soll, so liest die WfI die Datenreferenz aus dem WfMS aus und überträgt die Daten.

Datenzugriff

Für den Zugriff auf die Kooperationsdaten in DS_Q bestehen im wesentlichen zwei Alternativen: Zum einen kann eine API von DS_Q zum direkten lesenden Zugriff genutzt werden; zum anderen unterstützen viele Systeme eine Art Check-Out-Mechanismus. Dieser kann genutzt werden, um die Daten zunächst in ein handhabbares Format, beispielsweise XML, zu extrahieren und anschließend zu übertragen.

Eigentümer- und Besitzerverwaltung

Die Verwaltung der Eigentums- und Besitzerrechte muss die WfI als globale Instanz übernehmen. Dies allein reicht aber nicht aus. Wenn andere beteiligte Komponenten passende Unterstützung anbieten, dann sollte diese genutzt werden. So bieten die meisten PDMS auch Mechanismen für Computer-gestützte Gruppenarbeit an. Wird ein PDMS für die Datenhaltung verwendet, dann muss die WfI die internen Eigentums- und Besitzerrechte auf die unterliegenden Mechanismen abbilden. Dies kann von System zu System unterschiedlich sein.

4 Abbildung auf existierende Workflow- und EAI-Technolgie

Wie bereits deutlich wurde, findet die Integration unternehmensübergreifender Abläufe nicht „auf der grünen Wiese“ statt, sondern muss die Systemlandschaften auf den beteiligten Inseln berücksichtigen. Daher muss die WfI mit den vorhandenen Systemen verträglich sein. Weiterhin sollten nur dort Eigenentwicklungen stattfinden, wo keine passenden Komponenten am Markt erhältlich sind. In unserem Fall soll vorhandene EAI-Technologie eingesetzt werden, um notwendige neue Komponenten und vorhandene Systeme zusammen zu schalten. Wir streben die in Abbildung 2 gezeigte Grobarchitektur an.

Auf beiden Inseln arbeiten lokale WfMS Workflows nach den Vorgaben entsprechender Workflow-Typen ab. Wir gehen hier davon aus, dass die Anbindung an die WfI über speziell eingefügte Aktivitäten erfolgt. Diese werden im folgenden auch Datenflussaktivitäten genannt.

Die WfI besteht aus kommerziellen EAI-Brokern, wie beispielsweise IBM’s CrossWorlds [IBM02] oder Microsoft’s BizTalk [Ms01], und neu entwickelten Bausteinen, mit denen fehlende Funktionalität ergänzt wird. Im Bild ist ein Baustein für die Eigentümer-/Benutzerverwaltung vorgesehen.

Der EAI Broker kommuniziert mit den anderen Komponenten (Datenquellen, WfI-Komponenten, WfMS) über sogenannte Konnektoren. Diese haben drei Aufgaben:

- Wenn der EAI-Broker einen Dienst von einer integrierten Komponente anfordert, wird diese Anforderung vom Konnektor auf die Schnittstelle des entsprechenden Systems abgebildet.
- Wenn in einer angebundenen Komponente ein für die Integration relevantes Ereignis auftritt, wird dieses erkannt und an den EAI-Broker weitergeleitet. Die Ereigniserkennung erfordert entweder ein regelmäßiges Abfragen des Zustandes oder die Mithilfe der Komponente, beispielsweise über einen Callback-Mechanismus, einen direkten Aufruf oder, im Fall eines relationalen Datenbanksystems, durch einen geeignet definierten Trigger.
- Da EAI-Broker intern bevorzugt mit einem neutralen Datenformat arbeiten, bzw. unterschiedliche Komponenten mit unterschiedlichen Datenformaten arbeiten, führen die Konnektoren auch Datentransformationen durch.

Die Art und Weise, wie ein Integrationsmuster abgearbeitet werden soll, wird in so genannten *Kollaborationen* bestimmt. Wird über den Konnektor, der das WfMS anbindet, ein entsprechendes Ereignis signalisiert, wird im Broker der Start der entsprechenden Kollaboration ausgelöst.

Die beschriebene Architektur erlaubt uns, die in Abschnitt 2 eingeführten Kategorien von Datenflüssen zunächst unabhängig von der Technologie als Kollaborationen zu definieren und zu realisieren. Es ist Aufgabe der Konnektoren, die Abbildung auf konkrete Systeme durchzuführen. Zusätzlich benötigte Funktionalität kann ebenfalls leicht über neue Systemkomponenten integriert werden.

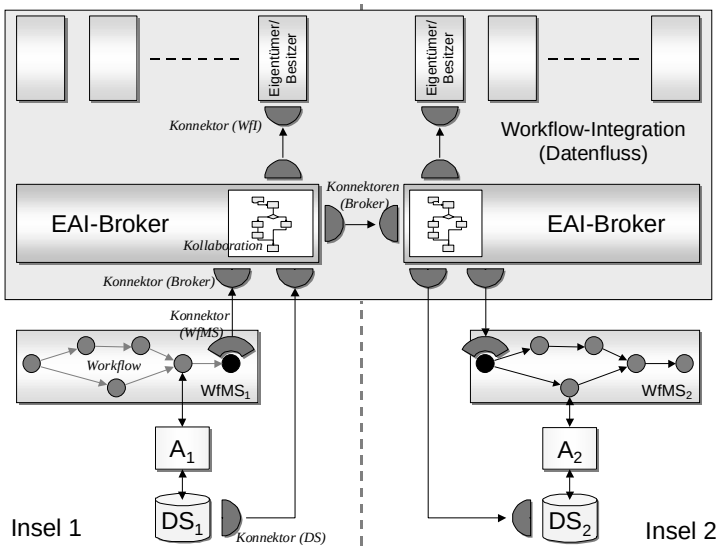


Abbildung 2: Integrationsarchitektur für Insel-übergreifende Datenflüsse

5 Verwandte Arbeiten

Wie in diesem Artikel deutlich geworden ist, streben wir eine Prozessintegration durch Bereitstellung einer Middleware-Komponente an, die auf Workflow-Technologie aufsetzt. Zur Realisierung dieser zentralen Komponente soll kein globales Workflow-Modell herangezogen werden, sondern es sollen die in der Realität auftretenden grundlegenden Abhängigkeiten zwischen Workflow-Umgebungen untersucht und eine Integrationskomponente entwickelt werden, welche die verschiedenen Formen dieser Abhängigkeiten spezifisch unterstützt. Unseres Wissens gibt es keine anderen Ansätze, die auch einen solchen Anspruch verfolgen.

Es gibt jedoch sehr wohl andere Ansätze, die sich mit der Thematik des firmenübergreifenden Austauschs von Produktdaten zum gemeinsamen Entwickeln von Produkten beschäftigen. Im Projekt ANICA wird versucht, über einen Integrationsbus den Zugriff auf verteilte CAX-Datenquellen zu ermöglichen. Dabei soll nicht nur der Ort der Daten transparent gehalten werden, sondern durch interne Konvertierung auch

(soweit möglich) das Format [AJSK98]. In dem in [AGL98] beschriebenen Projekt liegt der Fokus ebenfalls auf den Möglichkeiten der direkten Kopplung von heterogenen PDMS. Ein weiterer Ansatz, der Datenaustausch zwischen heterogenen PDMS ermöglichen soll, ist die „Product Data Markup Language“, die für das amerikanische Verteidigungsministerium (Departement of Defense, DoD) entwickelt wurde [Bur99]. Hier wird versucht, zu Integrationszwecken ein neutrales Integrationsschema zu nutzen. Somit gehören die vorgenannten Projekte zu einer Klasse von Forschungsansätzen, die das Gesamtproblem eher von der Seite der Datenintegration als von der Seite der Prozessintegration lösen wollen.

6 Zusammenfassung und Ausblick

Dieser Artikel leistet einen Beitrag zur Erforschung des Gebiets der unternehmensübergreifenden Workflows, die sich im wesentlichen über Abhängigkeiten zwischen heterogenen Workflow-Umgebungen (Inseln) definieren. Wir gehen dabei von einem Ansatz aus, der einerseits eine automatisierte Kontrolle solcher Abhängigkeiten ermöglicht, andererseits aber die betroffenen Systemumgebungen weitestgehend unbeeinflusst lässt.

In diesem Rahmen besteht der eigentliche Beitrag dieses Artikels in der Betrachtung von Datenflussabhängigkeiten, die im Anwendungsfall selbstverständlich zunächst erkannt und modelliert werden müssen, um sie dann zur Laufzeit automatisiert abwickeln zu können und damit einen reibungslosen Gesamtablauf, d.h. eine reibungslose ‚Kooperation’ heterogener Workflows, gewährleisten zu können. Als wesentliche Bestimmungsfaktoren haben wir die Art der Verwaltung der Kooperationsdaten auf der Quellseite, die Wirkung der Ausführung des eigentlichen Datenflusses auf sowohl die Quell- als auch die Zielinsel, den Modus der Bereitstellung der Kooperationsdaten auf der Zielinsel sowie die geeignete Regelung der Eigentums- und Besitzrechte angesichts der durch den eigentlichen Datenfluss sich verändernden Situation identifiziert.

Aus diesen Aspekten und den möglichen Kombinationen ihrer Ausprägungen ergeben sich eine Vielzahl von Integrationsmustern. Jedes dieser Muster beschreibt einen möglicherweise auftretenden Fall, so dass die Systemunterstützung bereits angesichts der Vielzahl dieser Muster sehr flexibel ausgelegt werden muss. Die integrierende Middleware muss auch hinsichtlich der heterogenen Systemlandschaften ausreichend flexibel einsetzbar sein, um die spezifischen Fähigkeiten der durch eine zu kontrollierende Datenflussabhängigkeit verbundenen Systeme (insbesondere Workflow-Management-Systeme) gezielt und gewinnbringend bei der automatischen Abwicklung zu berücksichtigen. Diese Anforderung macht es zunächst sehr schwer, die Systemlösung vorzuschlagen. Daher haben wir unseren Architekturvorschlag ausgehend von einem beispielhaften Integrationsmuster motiviert.

Die dargestellte Architektur, die auf den Einsatz existierender EAI-Technologie setzt, muss zwar noch verfeinert werden, um alle möglichen Fälle zu unterstützen, zeigt aber, wie von uns identifizierte Integrationsmuster als Kollaborationen allgemein anwendbar gemacht werden können. Systemspezifische Eigenschaften werden in Konnektoren gekapselt, welche die Systeme befähigen, an der Integration teilzunehmen. Fehlende Funktionalität wird zusätzlich durch ergänzenden Systemkomponenten integriert.

Weitere Untersuchungen und prototypische Implementierungen werden zeigen müssen, ob aktuelle EAI-Technologie geeignet ist, auch weitere Aspekte zu berücksichtigen. Offen ist auch, wie sich die EAI-Lösungen im Fehlerfall verhalten und inwieweit sich unterschiedliche EAI-Broker zur Zusammenarbeit bewegen lassen.

7 Literatur

[AGL98] Abramovici, M., Gerhard, D., Langenberg, L.: Supporting Distributed Product Development Processes with PDM, in: Krause, Heimann, Raupach.(Hrsg), New Tools and Workflows for Product Development, Proc. CIRP Seminar STC Design, Mai 1998, Berlin, Fraunhofer IRB Verlag, 1998, 1-11

[AJSK98] Arnold F., Janocha A. T., Swienczek B., Kilb T.: Die CAx-Integrationsarchitektur ANICA und ihre erste Umsetzung in die Praxis, in: Proc. Workshop Integration heterogener Softwaresysteme (IHS '98), 28. GI-Jahrestagung Informatik'98 - Informatik zwischen Bild und Sprache, Magdeburg, September 1998, 43-54

[BRZ00] Bon, M., Ritter, N., Zimmermann, J.: Interoperabilität heterogener Workflows, Proc. GI-Workshop Grundlagen von Datenbanken, 2000, 11-15

[BHR01] Bon, M., Härder, T., Ritter, N.: Produktdaten-Verwaltung in heterogenen Workflow-Umgebungen, Interner Bericht, Dezember 2001

[BRH02] Bon, M., Ritter, N., Härder, T.: Sharing Product Data among Heterogeneous Workflow Environments, in Proc. Int. Conf. CAD 2002 - Corporate Engineering Research, Dresden, März 2002, 139-149

[Bur99] Burkett, W.: PDML - Product Data Markup Language - A New Paradigm for Product Data Exchange and Integration, 30.04.1999, www.pdml.org/whitepap.pdf

[IBM02] Technical Introduction to IBM CrossWorlds, IBM Corporation 2002

[KRS01] Kulendik, O., Rothermel, K., Siebert, R.: Cross-organizational workflow management - General Approaches and their Suitability for Engineering Processes. in: Schmid, B., Stanoevska-Slabeva, K., Tschammer, V. (Hrsg.): Proc. First IFIP-Conference on E-Commerce, E-Business, E-Government: I3E 2001, Zürich, Schweiz, Oktober 2001

[LR00] Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques, Prentice Hall PTR (ECS Professional), 2000, ISBN 0-13-021753-0

[Ms01] Microsoft BizTalk Server 2000: Documented, Microsoft Press, 2001

Realisierung eines adaptiven Replikationsmanagers mittels J2EE-Technologie

Heiko Niemann[1], Wilhelm Hasselbring[2], Michael Hülsmann[2], Oliver Theel[2]

1. OFFIS
   Escherweg 2
   26121 Oldenburg
   heiko.niemann@offis.de
2. Carl von Ossietzky Universität Oldenburg
   Fachbereich Informatik
   26111 Oldenburg
   {hasselbring|michael.huelsmann|theel}
   @informatik.uni-oldenburg.de

Kurzfassung: Die Integration in heterogenen Systemlandschaften bedeutet häufig eine Kopplung zur Replikation von Daten lokaler, autonomer Systeme. Dabei soll einerseits Datenmanipulation unter Wahrung der Konsistenz erfolgen, andererseits sollen die beteiligten Systeme i.A. ihre Autonomie beibehalten. Dieser Konflikt der Anforderungen wird durch bekannte Replikationsverfahren nur begrenzt gelöst: Je nach Strategie wird ein Kompromiss zwischen Konsistenz und Autonomie eingegangen. Hier kann ein konfigurierbarer, adaptiver Replikationsmanager Abhilfe schaffen, der eine Kombination bildet aus synchroner Replikation, die eine hohe Konsistenz bei eingeschränkter Verfügbarkeit garantiert, und asynchroner Replikation, die bei eingeschränkter Konsistenz den Systemen eine größere Autonomie belässt. Die Konfiguration des Replikationsmanagers wird durch ein Regelsystem gesteuert, dass Faktoren wie Tageszeit oder Performanz berücksichtigt. Im vorliegenden Papier wird eine Realisierung des adaptiven Replikationsmanagers auf Basis der J2EE-Technologie vorgestellt. Sie bietet sich an, da hier z.B. Transaktions- und Nachrichtendienste genutzt werden können sowie Möglichkeiten zur technischen Anbindung der beteiligten Systeme zur Verfügung stehen.

1 Einleitung

Gegenüber einer Datenreplikation in einem homogenen Systemumfeld, in dem z.B. die Replikation von gleichen Tabellen einer relationalen Datenbank vom Datenbankmanagementsystem selbst übernommen wird, stellt die Replikation in einem heterogenen, autonomen Systemumfeld höhere Ansprüche an die Umsetzung einer geeigneten Strategie. Hier werden zusätzlich Fragen wie die Interoperabilität der Systeme, lokale Schemata oder Semantik der Daten aufgeworfen. Nachdem die zu replizierenden Daten und die involvierten Speichersysteme identifiziert wurden, sind folgende Aufgaben von einer Datenreplikation in diesem Kontext zu bewältigen:

1. Bestimmung einer Replikationsstrategie
2. Auswahl der Transaktionsverarbeitung
3. Technische Anbindung der beteiligten Systeme
4. Integration der lokalen Schemata

Der hier vorgestellte *adaptive Replikationsmanager (ARM)* dient zur Umsetzung dieser vier Aufgaben. Ein Schwerpunkt des ARM bildet naturgemäß die Replikationsstrategie (siehe Punkt 1), die auf Grund unterschiedlicher Konsistenz- und Autonomieanforderungen der heterogenen, autonomen Systeme *konfigurierbar* und *adaptiv* sein soll. Damit einergehend sind die eingesetzten Transaktionskonzepte und die technische Anbindung der beteiligten Systeme relevant. Die J2EE-Technologie unterstützt gerade diese Punkte 2) und 3) durch die entsprechenden Dienste innerhalb einer einheitlichen

Architektur. Da die beteiligten Systeme in einer heterogenen Systemlandschaft i.A. über unterschiedliche Schemata verfügen, ist eine Schemaintegration nötig (siehe Punkt 4). Beim Vorgang der Schemaintegration werden eher semantische Fragen berücksichtigt [SK92], die wir im vorliegenden Papier nicht behandeln.

In Abschnitt 2 werden Replikationsverfahren und Transaktionskonzepte als Grundlagen vorgestellt. Anschließend wird der Aufbau und die Funktionalität des ARM beschrieben: Die Architektur des ARM wird in Abschnitt 3 gezeigt. Abschnitt 4 diskutiert eine Implementierung auf Basis der J2EE-Technologie. In Abschnitt 5 wird das Regelsystem vorgestellt, das der Arm für seine Arbeitsweise (siehe Abschnitt 6) nutzt. Die einzelnen Komponenten des ARM zeigt Abschnitt 7. Abschließend werden in Abschnitt 8 verwandte Arbeiten diskutiert und in Abschnitt 9 ein kurzes Fazit gezogen.

2 Grundlagen

2.1 Replikationsverfahren

In [BD96] werden Klassifizierungskriterien für Replikationsverfahren, die üblicherweise benutzt werden, anhand von drei Fragestellungen diskutiert:

1. Wie erfolgt die replikationsübergreifende Synchronisation der Zugriffsoperationen, d.h. welches Korrektheitskriterium wird verwendet? Dies führt zu *syntaktischen* bzw. *semantischen* Verfahren.
2. Wie werden die Replikate aktualisiert? Es können *synchrone* bzw. *asynchrone* Verfahren unterschieden werden.
3. Wie sieht die Konsistenzsicherung im Fehlerfall aus, z.B. bei Netzpartitionierung? Diese Frage erlaubt eine Einteilung in *pessimistische* bzw. *optimistische* Verfahren.

Das bekannteste Korrektheitskriterium ist wohl die *One-Copy-Serialisierbarkeit (1SR*, siehe Punkt 1): Eine nebenläufige Ausführung von verteilten Transaktionen ist *one-copy-serialisierbar*, wenn sie mit einer seriellen Ausführung dieser Transaktionen auf einer nicht replizierten Datenbank äquivalent ist. In diesem Sinne soll unter *Konsistenz* verstanden werden, dass eine Replikationsstrategie nach einem Korrektheitskriterium wie z.B. 1SR arbeitet. Die Replikationsstrategien, die ein serialisierbares Korrektheitskriterium verwenden, werden den *syntaktischen* Verfahren zugeordnet. Bei den *semantischen* Verfahren wird Wissen über die Semantik der Zugriffsoperationen bzw. Transaktionen zusätzlich ausgenutzt.

Falls z.B. durch Netzpartitionierung (siehe Punkt 3) nicht mehr alle Replikate erreicht werden können, kann ein Replikationsverfahren entweder die Verfügbarkeit der Daten zu Gunsten der Konsistenz einschränken (*pessimistische Verfahren*) oder eine Weiterverarbeitung zulassen und erst einmal Inkonsistenzen in Kauf nehmen (*optimistische Verfahren*). Diese Inkonsistenzen werden auch als *Konflikte* bezeichnet, wobei *Lesekonflikte* (Lesen veralteter Daten) und *Änderungskonflikte* (zwei Systeme ändern den gleichen Wert eines replizierten Objekts jeweils lokal, d.h. ihr eigenes Replikat) auftreten können. Bei der Zusammenführung müssen die Konflikte erkannt und behandelt werden.

Auf Punkt 2 der Klassifizierungsmöglichkeiten wird in [BD96] nicht weiter eingegangen. In diesem Papier wollen wir uns jedoch auf diesen Punkt konzentrieren. Es soll analog zur Kommunikation von Systemen zwischen *synchroner* und *asynchroner* Replikation unterschieden werden, um einen Fokus auf die Interaktion der Systeme zu legen:

- Synchrone Replikation bei hoher Konsistenzanforderung
- Asynchrone Replikation bei hoher Autonomieanforderung

***Synchrone Replikation*:** Mit synchroner Replikation ist die kontrollierte Änderung der notwendigen Replikate innerhalb einer Transaktion gemeint. Für das ROWA-Verfahren (Read One, Write All) [TG82] sind das alle Replikate, für die Voting-Verfahren (siehe z.B. [Gi79]) sind das die Replikate, die zu einem Quorum gehören. Synchrone Replikation genügt hohen Konsistenzanforderungen, aber es müssen Einschränkungen hinsichtlich Autonomie, d.h. Verfügbarkeit und Performanz, hingenommen werden. Wir werden hier auch quasi-synchrone Replikation (Abschnitt 2.2) mit parallelen Sagas betrachten.

***Asynchrone Replikation*:** Bei der asynchronen Replikation werden die Replikate zeitversetzt aktualisiert. Beim Peer-To-Peer-Verfahren (siehe z.B. [GRR98]) können alle Kopien geändert werden, die Aktualisierung wird asynchron vorgenommen und es werden somit bewusst Konflikte in Kauf genommen. Bei der Zusammenführung müssen dann die möglicherweise aufgetretenen Konflikte geeignet behandelt werden. Das Peer-To-Peer-Verfahren belässt den Systemen eine hohe Autonomie, da nur geringe Einschränkungen für das Bereitstellen des Propagierungsauftrags auftreten. Es nimmt aber Einschränkungen hinsichtlich der Konsistenz in Kauf, d.h. das Korrektheitskriterium 1SR wird nicht erfüllt.

2.2 Transaktionskonzepte

Um einen ordnungsgemäßen Dienst zu gewährleisten, müssen die Replikationsverfahren nach den Prinzipien der Transaktionsverarbeitung implementiert werden (zu verschiedenen Konzepten siehe z.B. [GR93]). Im Folgenden soll dargestellt werden, welche Transaktionskonzepte für uns in Frage kommen:

- 2PC- / XA-Protokoll [Gr78] für die synchrone Replikation XA-fähiger Systeme.
- Sagas [GS87] für die *quasi-synchrone* Replikation nicht XA-fähiger Systeme.
- Queued Transactions [BN97] für die asynchrone Replikation.

***2-Phasen-Commit-(2PC-)Protokoll/XA-Protokoll*:** Das 2PC-Protokoll basiert auf dem Grundgedanken, dass alle an einer globalen Transaktion T beteiligten Systeme sich darauf einigen, ob T durchgeführt oder abgebrochen wird [Da96]. Die X/Open, ein Konsortium von DBMS-Herstellern, hat Systemkomponenten und Schnittstellen für verteilte Transaktionsverarbeitung spezifiziert, die auf dem 2PC-Protokoll fußt. Die lokalen Systeme stellen einen Ressourcenmanager bereit, der das sogenannte XA-Protokoll unterstützt. Hierüber steuert ein globaler Transaktionsmanager die verteilte Transaktion. Das XA-Protokoll wird von vielen Systemherstellern unterstützt, so dass diese Form der Verarbeitung zu einem defacto-Standard geworden ist.

Sagas ([GS87], auch „verkettete Transaktion“) gehören zu den offen-geschachtelten Transaktionen. Eine *Saga-Transaktion* besteht aus einer Folge von einzelnen Transaktionen $T_1,...,T_n$, die jeweils bei erfolgreicher Beendigung „committed“ werden, d.h. ihre Änderungen sichtbar machen. Für Sagas treffen isolierte Zurücksetzbarkeit und Serialisierbarkeit im Sinne der ACID-Eigenschaften nicht mehr zu. In unserem Fall sollen Sagas genutzt werden, um nicht XA-fähige Systeme an der Replikation teilnehmen zu lassen. Durch Überlappung der Einzeltransaktionen T_i (siehe Parallel Sagas [GS87]) kann eine scheinbar zeitgleiche Verarbeitung erzielt werden, d.h. eine quasi-synchrone, aber nicht echt synchronisierte Verarbeitung. Die Aktualisierung der Replikate kann so erfolgen, dass in jeder Einzeltransaktion T_i jeweils genau ein Replikat geändert wird. In jeder Einzeltransaktion T_i wird somit eine lokale, „flache“ Transaktion ausgeführt. Diese spezielle Variante des Sagas-Konzepts wollen wir im weiteren durch den Index R kennzeichnen, also $\boldsymbol{Sagas_R}$. Bei Abbruch einer Einzeltransaktion T_i wird gemäß dem Forward Recovery neu aufgesetzt

Queued Transactions: In [BN97] wird das Queued Transaction Processing Modell anhand des Client/Server-Modells erläutert. Dabei zerfällt die globale Transaktion in drei Teiltransaktionen, die jeweils den ACID-Eigenschaften unterliegen. T_1: Ein Client stellt eine Anfrage in die *request queue*. T_2: Der Server bearbeitet die Anfrage und erteilt eine Antwort in die *reply queue*. T_3: Der Client holt die Antwort und bearbeitet sie. Die Queued Transactions können wie folgt für die asynchrone Replikation verwendet werden (hier als $\boldsymbol{QT_R}$ bezeichnet): Ein Client stellt eine *Replikationsanforderung*, d.h. eine Änderungsanforderung aller Replikate in eine „Replica Queue". Ein Server holt die Anforderung aus der Replica Queue und aktualisiert die Replikate. Auf das Schreiben einer Antwort und die folgende Bearbeitung der Antwort kann verzichtet werden. Im fehlerfreien Fall ist die Transaktion erfolgreich abgeschlossen, im Fehlerfall wird ein Konfliktmanagement für die weiteren Schritte aktiviert.

3 Architektur des adaptiven Replikationsmanagers

Um sowohl die Vorteile der synchronen Replikation (Konsistenz) als auch der asynchronen Replikation (Autonomie) zu nutzen, ist eine Kombination der jeweiligen Verfahren nötig. In Abhängigkeit der Replikationsanforderungen und der Zustände der beteiligten Systeme kann zwischen den Verfahren gewechselt bzw. ein Mischverfahren verwendet werden, wobei eine dynamische Anpassung an veränderte Bedingungen erfolgen soll. Dabei werden grundsätzlich Lesezugriffe auf veraltete Daten gemäß der Peer-To-Peer-Replikation toleriert. Eine derartige *adaptive Replikationsstrategie* [NH02] kann von einem ARM (adaptiver Replikationsmanager) umgesetzt werden, der gemäß der in Abbildung 1 gezeigten Architektur aufgebaut ist. Die ARM realisiert durch die transaktionsgesicherte Verarbeitung aller Replikationsanforderungen ein Verfahren nach dem Korrektheitskriterium *Konvergenz* [Le97]. Konvergenz besagt, dass alle Replikate eines replizierten Objekts letztendlich den gleichen Wert annehmen und dieser Wert derjenige ist, der durch die letzte, abgeschlossene Transaktion geschrieben wurde.

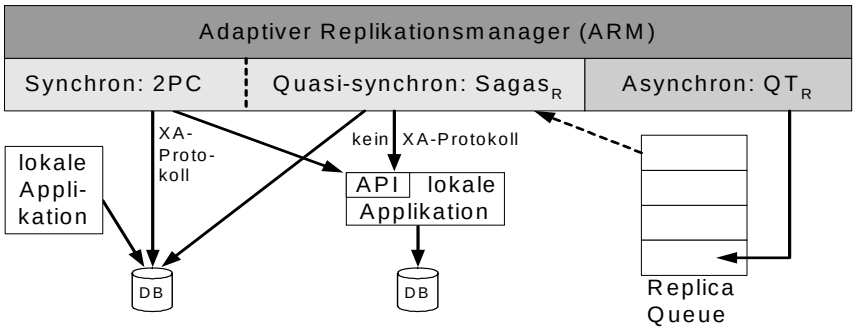


Abbildung 1 Architektur des konfigurierbaren, adaptiven Replikationsmanagers

Der ARM verarbeitet Replikationsanforderungen anhand eines parametrisierten Regelsystems (Abschnitt 5), wobei die Konfiguration dynamisch den aktuellen Gegebenheiten angepasst wird. Auf Grund der aktuellen Einstellungen werden diejenigen Systeme bestimmt, die synchron bzw. asynchron aktualisiert werden (Abschnitt 6). Die synchrone Replikation wird transaktional entweder gemäss dem 2PC durchgeführt, sofern die beteiligten Systeme das XA-Protokoll unterstützen, oder gemäss dem Sagas$_R$-Konzept, sofern das XA-Protokoll nicht unterstützt wird (letzteres bedeutet eine quasi-synchrone Replikation). Die asynchrone Replikation nach dem QT$_R$-Konzept geschieht dadurch, dass die

Replikationsanforderung je Replikat zunächst in eine „Replica Queue“ eingefügt wird. Zu einem späteren Zeitpunkt wird die zurückgestellte Anforderung gemäß dem Prinzip verteilter Transaktionen bearbeitet, d.h. erstens die Anforderung aus der Replica Queue holen und zweitens ein beteiligtes System aktualisieren. Hierfür wird wiederum der Mechanismus der synchronen Transaktionsverarbeitung genutzt (gestrichelter Pfeil in Abbildung 1).

Der Zugriff auf die operativen Systeme hängt davon ab, ob auf die Datenquelle direkt oder über Schnittstellen zugegriffen wird, d.h. ob mit einem direkten Datenbankzugriff die Aktualisierung erfolgen kann oder ob die Datenänderung mittels eines entfernten Prozeduraufrufes auf die Schnittstellen (APIs) einer lokalen Anwendung geschieht. In [RMB00] wird zwischen *Datenbankintegration*, d.h. ein Zugriff auf die Datenquelle erfolgt parallel zu den lokalen Anwendungen, und *Applikationsintegration*, d.h. ein Zugriff erfolgt über die Schnittstellen der lokalen Anwendungen, unterschieden. Letzteres ist immer dann erforderlich, wenn keine direkte Datenmanipulation an der zugrunde liegenden Datenquelle erlaubt ist oder diese zu komplex ist.

4 Realisierung mittels J2EE-Technologie

Die J2EE-Technologie (Java 2 Enterprise Edition [JNB01]) bietet sich aus folgenden Gründen für die Realisierung des ARM an:

- Sie umfasst einen Anwendungsserver mit Unterstützung für synchrone und asynchrone Kommunikation: Ein Enterprise JavaBeans Server (EJB-Server) verwaltet und unterstützt die EJB-Anwendungen durch sogenannte Container für EJBs. Verschiedene Dienste, wie z.B. ein Transaktionsdienst (Java Transaction Service, JTS) und ein Nachrichtendienst (Java Message Service, JMS), können bei Bedarf eingebunden werden.
- Sie ermöglicht Zugriffe auf operative Systeme (Legacy-Systeme): Für Zugriffe auf Datenbanken steht die Java Database Connectivity (JDBC) zur Verfügung. Die Java Connector Architecture (JCA) spezifiziert Zugriffe auf Anwendungen bzw. deren Schnittstellen.
- Sie unterstützt Nutzungs-Schnittstellen und -Programme: Sowohl mit der Java 2 Standard Edition (J2SE) als auch der Java 2 Micro Edition (J2ME) können Client-Anwendungen entwickelt werden. Java Server Pages (JSP) bzw. Java Servlets unterstützen die Entwicklung von Web-Anwendungen. Die Realisierung der Client-Anwendungen wird im vorliegenden Papier nicht betrachtet.

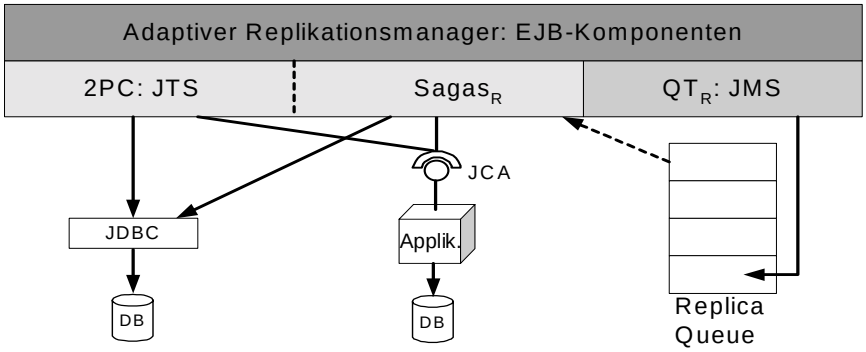


Abbildung 2 Realisierung des adaptiven Replikationsmanagers mittels J2EE-Technologie

Der ARM in Abbildung 2 wird als EJB-Anwendung realisiert und kommt auf einem J2EE-Server zum Einsatz (Deployment). Die Umsetzung der Transaktionsverarbeitung im synchronen Fall erfolgt über JTS (inkl. XA-Protokoll). Falls das $Sagas_R$-Konzept verwendet werden muss, ist eine eigene Implementierung nötig. Das QT_R-Konzept für den asynchronen Fall wird über den Nachrichtendienst JMS (inkl. XA-Protokoll) realisiert, wobei die Replica Queue als Topic (Publish/Subscribe-Verfahren) oder Queue (Point-To-Point-Verfahren) gemäß der JMS-Terminologie gestaltet wird [MC01].

Die operativen Systeme werden im Fall einer Datenbankintegration über JDBC [SS00] angebunden, im Fall einer Applikationsintegration erfolgt der Zugriff gemäss JCA (siehe z.B. [SSN02]). Im letzteren Fall wird für das betreffende System ein Ressourcenadapter benötigt, über den die individuellen Schnittstellen angesprochen werden. Sowohl über JDBC als auch JCA können XA-fähige und nicht XA-fähige Ressourcen angebunden werden. Clients, die z.B. in J2SE oder J2ME implementiert sind, können Replikationsanforderungen über RMI (Remote Methode Invocation) an den ARM übergeben.

5 Das Regelsystem des adaptiven Replikationsmanagers

Wir definieren zunächst den Begriff der sogenannten *Replikationseinheit (RE)*: Eine Replikationseinheit ist entweder ein System, das eine Replikationsanforderung stellt, eine Relation oder ein Tupel.

Die hier vorgestellte adaptive Replikationsstrategie kombiniert die synchrone und asynchrone Replikationsstrategie wie folgt: Zunächst wird von synchroner Replikation ausgegangen, um eine hohe Konsistenz zu erreichen. Zwischen synchroner und quasi-synchroner Replikation wird hier nicht unterschieden, weil dadurch nur die technische Anbindung der Systeme beeinflusst wird. Falls ein oder mehrere Systeme die Autonomie der anderen gefährden bzw. zu stark einschränken, wird für diese Systeme dynamisch zur asynchronen Replikation gewechselt. Nach einer Prüfung oder einem festgelegtem Zeitintervall wird wieder zur synchronen Replikation zurückgewechselt. Der Wechsel zwischen synchroner und asynchroner Replikation ist konfigurierbar, d.h. dem Replikationsmanager werden Regeln vorgegeben, anhand derer er automatisch Anpassungen an die Replikationsstrategie vornimmt.

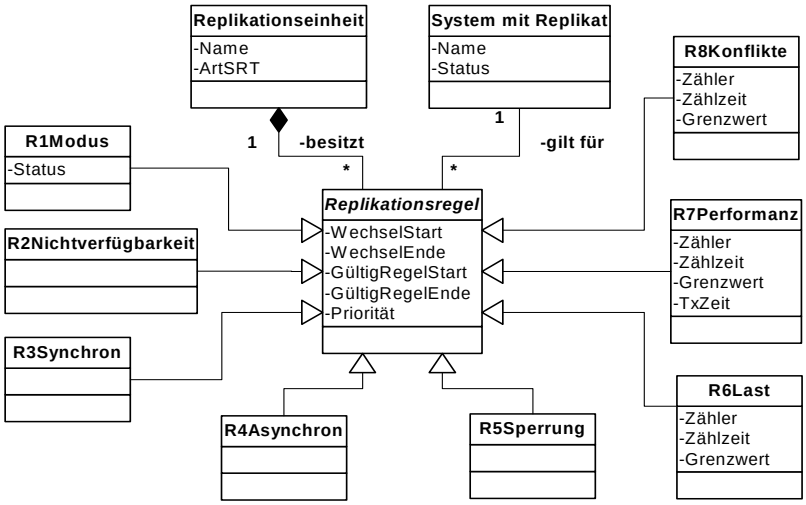


Abbildung 3 vereinfachtes Klassendiagramm des Regelsystems

Je RE können Regeln vergeben werden. Eine Regel gilt genau für ein System mit Replikat. Für jedes System mit Replikat gibt es eine ausgezeichnete Regel, die besagt, ob das System grundsätzlich synchron aktualisiert wird (z.B. ein „Mastersystem“), grundsätzlich asynchron aktualisiert wird (z.B. ein Data Warehouse) oder wechselfähig ist.

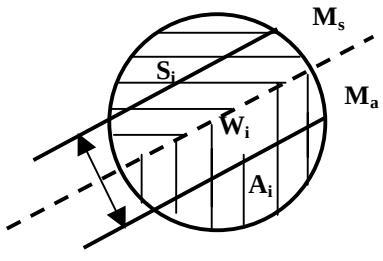
Das Regelsystem in Abbildung 3 als UML-Klassendiagramm [RJB99] ist wie folgt definiert: Anhand des Attributes ArtSRT wird für eine RE festgehalten, ob es sich um ein System, eine Relation oder ein Tupel handelt. Auch die Spezifikation einer Default-Einstellung ist möglich, da nicht für alle Systeme bzw. Relationen individuelle Regeln vorgegeben werden müssen. Eine RE besitzt mehrere Replikationsregeln (abstrakte Klasse). Eine Replikationsregel gilt für Systeme mit Replikat und kann zu speziellen Regeln abgeleitet werden. Diese speziellen Regeln legen fest, wann zwischen synchroner und asynchroner Replikation gewechselt wird bzw. ob für ein System grundsätzlich synchron oder asynchron repliziert werden soll. Die speziellen Regeln lauten (siehe auch Abbildung 3):

- R1 Das System (Attribut Status) wird synchron bzw. asynchron aktualisiert oder es ist wechselfähig.
- R2 Asynchrone Replikation bei Nichtverfügbarkeit.
- R3 Synchrone Replikation im gegebenen Zeitintervall.
- R4 Asynchrone Replikation im gegebenen Zeitintervall.
- R5 Sperrung im gegebenen Zeitintervall. Die Anforderung wird zwar angenommen, aber erst nach Ende des Zeitintervalls asynchron ausgeführt.
- R6 Wechsel zur asynchronen Replikation, wenn in einem gegebenen Zeitintervall zu viele Replikationsanforderungen auftreten.
- R7 Wechsel zur asynchronen Replikation, wenn in einem gegebenen Zeitintervall die Performanz zu Wünschen übrig lässt.
- R8 Synchrone Replikation im gegebenen Zeitintervall wegen zu großer Anzahl von Konflikten.

Die Regeln R2 bis R8 können je System mehrfach gesetzt werden, z.B. eine Sperrung gemäß R5 von 10:00 Uhr bis 11:30 Uhr und 15:00 Uhr bis 16:15 Uhr. Da die Regeln R3 und R8 einen Wechsel von asynchron zu synchron begründen, die übrigen Regeln einen Wechsel von synchron zu asynchron, können Konflikte innerhalb der Regeln auftreten. Durch Vergabe von Prioritäten kann festgelegt werden, welche Regel den Vorzug erhält.

6 Die Arbeitsweise des adaptiven Replikationsmanagers

Die Abbildung 4 zeigt, wie der ARM die Systeme mit Replikat in die Gruppen synchron und asynchron zu aktualisierender Systeme einteilt:



$$S_i \subseteq M_s \subseteq (S_i \cup W_i)$$
$$A_i \subseteq M_a \subseteq (A_i \cup W_i)$$

Abbildung 4 Aufteilung der Systeme mit Replikat

Die Menge **I** sei die Menge der REen und die Default-Einstellung, **M** sei die Menge der Systeme mit Replikat, $\mathbf{S_i}$ die Menge der Systeme, die synchron aktualisiert werden, $\mathbf{A_i}$

die Menge der Systeme, die asynchron aktualisiert werden und $\mathbf{W_i}$ die Menge der wechselfähigen Systeme, mit $\mathbf{i \in I}$. S_i, A_i und W_i partitionieren M.

Wenn eine Replikationsanforderung gestellt wird, teilt der ARM anhand der Regeln und der aktuellen Situation die beteiligten Systeme in zwei Mengen (siehe Abbildung 4): $\mathbf{M_s}$ sei die Menge der Systeme, die synchron aktualisiert werden, und $\mathbf{M_a}$ sei die Menge der Systeme, die asynchron aktualisiert werden. M wird durch M_s und M_a partitioniert. M_a und M_s werden für jede Replikationsanforderung neu bestimmt und können somit variieren (in Abbildung 4 durch den Doppelpfeil dargestellt). Zu einem bestimmten Zeitpunkt hängen M_a und M_s einerseits vom Regelsystem (Konfiguration) und andererseits vom sich dynamisch ändernden System ab (Adaption).

7 Die Komponenten des adaptiven Replikationsmanagers

Der ARM kann in einzelne Komponenten gegliedert werden, die mehr oder weniger unabhängig bestimmte Aufgaben bewältigen. In Abbildung 5 ist dargestellt, aus welchen Komponenten (gemäß der J2EE-Terminologie EJB-Komponenten) der ARM besteht.

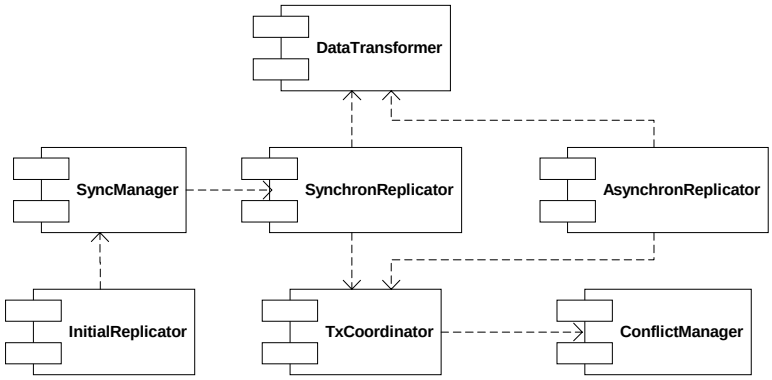


Abbildung 5 Komponenten des ARM, dargestellt als UML-Komponenten-Diagramm [RJB99]

Die einzelnen Komponenten erfüllen folgende Aufgaben:

- Der SynchronReplicator nimmt Replikationsanforderungen entgegen, bearbeitet sie und liefert einen Rückgabewert. Die Bearbeitung umfasst die synchrone Replikation sowie das Einstellen in die Replica Queue.
- Der AsynchronReplicator verwaltet die Replica Queue, d.h. die asynchronen Replikationsaufträge werden für die jeweiligen Systeme ausgeführt.
- Der DataTransformer ist für die Transformation von Daten zum globalen und zu den lokalen Datenschemata im Sinne der Schemaintegration verantwortlich.
- Der TxCoordinator steuert die transaktionale Verarbeitung des ARM mittels des Transaktionssystems des J2EE-Servers. Für das $Sagas_R$-Konzept wird eine eigene Implementierung benötigt. Konflikte werden an den ConflictManager übergeben.
- Der ConflictManager ist sowohl für die Erkennung von Konflikten als auch für die Behandlung der aufgetretenen Konflikte zuständig.
- Der SyncManager synchronisiert ein anfragendes System, z.B. ein mobiles Gerät, mit den zentralen Systemen.
- Der InitialReplicator dient der erstmaligen Erstellung einer replizierten Datenquelle oder der Aktualisierung von „Downloads“ der mobilen Geräten.

Nur der SynchronReplicator kann einen Wechsel von synchron zu asynchron bewirken. Der AsynchronReplicator sorgt ausschließlich für den Wechsel von asynchron zu synchron, wobei der Wechsel nur dann erfolgen kann, wenn für das betreffende System keine Aufträge in der Replica Queue anstehen.

8 Verwandte Arbeiten

Zum Thema Replikation sind eine Vielzahl von Arbeiten erstellt worden, so dass an dieser Stelle nur ein kleiner Ausschnitt betrachtet werden soll. Einen Schwerpunkt auf Autonomie setzen die optimistischen Verfahren. Gerade hier finden neuerdings auch mobile Geräte Berücksichtigung. Vertreter von Systemen, die nach diesen Konzepten entwickelt wurden, sind CODA [Sa02], Bayou [TTP95] oder Rumor [GRR98].

In [LH00] wird eine konfigurierbare Replikation speziell bei mobilen Applikationen diskutiert. Lenz [Le97] stellt in seiner Dissertation eine adaptive Datenreplikation vor, die pessimistische und optimistische Ansätze kombiniert. Es werden sogenannte Konsistenzinseln gebildet, deren Replikate alle den gleichen, aktuellen Wert haben. Nicht zur Konsistenzinsel gehörende Replikate können zeitverzögert aktualisiert werden.

Fragen zum Konfliktmanagement und zum Datenabgleich (reconciliation) tauchen bei Netzpartitionierung auf. Zur Schreib/Schreib-Konflikterkennung wird ein klassisches Verfahren in [PPR83] vorgestellt. In diesem optimistisch-syntaktischen Ansatz sollen Inkonsistenzen am replizierten Datenbestand erkannt und lokalisiert werden. Ein weiteres klassisches Verfahren ist das sogenannte optimistische Protokoll [Da84]. Es pflegt einen sogenannten precedence graph, mittels dem partitionsbezogene Konflikte erkannt und beseitigt werden können. In [GAB83] werden diese Probleme bei der Zusammenführung dadurch gelöst, dass im Datensatz bestimmte Regeln abgelegt werden, die bei Konflikten eine Behandlung ermöglichen.

In [Ha97] wird die Replikation über die Schemata föderierter Datenbanksysteme erläutert, d.h. hier wurde insbesondere ein Schwerpunkt auf die verschiedenen Schemata in verteilten Systemen gelegt.

9 Zusammenfassung und Ausblick

In heterogenen, autonomen Informationssystemen kann die Replikation von Daten durch rein synchrone bzw. asynchrone Verfahren nicht optimal gelöst werden. Ein adaptiver Replikationsmanager (ARM), der durch Kombination der beiden Varianten eine konfigurierbare, adaptive Replikationsstrategie implementiert, kann eine optimalere Lösung erreichen, insbesondere dann, wenn zusätzlich Funktionalität wie z.B. das Konfliktmanagement unterstützt wird. Diese Vorteile werden durch die Abhängigkeit vom ARM bzw. dem System, auf dem der ARM installiert ist, erkauft. Eine Unabhängigkeit von einem zentralen System kann durch Clustern vom ARM erreicht werden. Für die Implementierung bietet sich die Java-Technologie J2EE mit den entsprechenden Diensten an.

In diesem Beitrag wird die Architektur des ARM sowie eine Realisierung auf Basis der J2EE-Technologie gezeigt. Die Umsetzung der adaptiven Replikationsstrategie wird vom ARM über ein hier vorgestelltes Regelsystem gewährleistet. Weiterhin werden die einzelnen Komponenten und Funktionalitäten des ARM betrachtet, sowie die technische Anbindung der beteiligten Systeme diskutiert.

Ein explorativer Prototyp zur Validierung der Ergebnisse wurde auf Basis des J2EE-Servers Bea Weblogic (www.bea.com) implementiert. Als Systeme mit Replikat wurden

die XA-fähigen Datenbanken Cloudscape (www.ibm.com), Oracle (www.oracle.com) und MS-SQL-Server (www.microsoft.com) mit identischem Schema eingesetzt. Des weiteren wurde eine Anbindung des nicht XA-fähigen SAP R/3 Systems (www.sap.com) über einen Ressourcenadapter gemäß der Java Connector Architecture (JCA) untersucht. Zukünftige Arbeitspakete beinhalten eine genauere Analyse des Regelsystems und formale Korrektheitsprüfungen, z.B. zur Deadlock- und Lifelock-Erkennung, sowie ein Monitoring der Arbeitsweise des ARM. Als Anwendungsbereich werden Krankenhausinformationssysteme betrachtet [NHW02].

Literaturverzeichnis

[BD96] Beuter, T.; Dadam, P.: Prinzipien der Replikationskontrolle in verteilten Systemen. In: Informatik Forschung und Entwicklung 11 (4), 1996, S.203-212.

[BN97] Bernstein, P.; Newcomer, E.: Principles of Transaction Processing. Morgan Kaufmann, 1997.

[Da84] Davidson, S.: Optimism and Consistency in Partitioned Distributed Database Systems. In: ACM Transactions on Computer Systems 9 (3), 1984, S.456-481.

[Da96] Dadam, P.: Verteilte Datenbanken und Client/Server-Systeme. Springer-Verlag, 1996.

[GAB83] Garcia-Molina, H.; Allen, T.; Blaustein, B. T. et al.: Data-Patch: Integrating Inconsistent Copies of a Database After a Partition. In:Symposium on Reliability in Distributed Software and Database Systems, Clearwater Beach, FL, 1983 S.38-44.

[Gi79] Gifford, D. K.: Weighted Voting for Replicated Data. In:ACM Symposium on Operating Systems Principles (SIGOPS), Pacific Grove, California, 1979 S.150-159.

[Gr78] Gray, J.: Notes on Database Operating Systems. In R. Bayer et al. (ed): Operating Systems: an Advanced Course, Heidelberg, Springer Lecture Notes in Computer Science, 1978.

[GR93] Gray, J.; Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.

[GRR98] Guy, R.; Reiher, P.; Gunter, M. et al.: Rumor: Mobile Data Access Through Optimistic Peer-To-Peer Replication. In:International Conference on Conceptual Modeling (ER), Singapore, 1998 S.254-265.

[GS87] Garcia-Molina, H.; Salem, K.: Sagas. In:ACM SIGMOD, San Francisco, 1987 S.249-260.

[Ha97] Hasselbring, W.: Federated integration of replicated information within hospitals. In: International Journal on Digital Libraries 1 (3), 1997, S.192-208.

[JNB01] Juric, M.; Nagappan, R.; Basha, J. et al.: Professional J2EE EAI. WROX, 2001.

[Le97] Lenz, R.: Adaptive Datenreplikation in verteilten Systemen. Teubner, Leipzig 1997.

[LH00] Lubinski, A.; Heuer, A.: Configured Replication for Mobile Applications. In:Proc. of Baltic DB & IS, Vilnius, Litauen, 2000 S.1-5.

[MC01] Monson-Haefel, R.; Chappel, D. A.: Java Message Service. O'Reilly & Associates, 2001.

[NH02] Niemann, H.; Hasselbring, W.: Adaptive Replikationsstrategie für heterogene, autonome Informationssysteme. In:14. Workshop über Grundlagen von Datenbanken, Fischland, 2002 S.81-85.

[NHW02] Niemann, H.; Hasselbring, W.; Wendt, T. et al.: Kopplungsstrategien für Anwendungssysteme im Krankenhaus. In: Wirtschaftsinformatik 44 (5), 2002, S.425-434.

[PPR83] Parker, D. S.; Popek, G.; Rudisin, G.: Detection of Mutual Inconsistency in Distributed Systems. In: IEEE Transactions on Software Engineering 9 (3), 1983, S.240-247.

[RJB99] Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

[RMB00] Ruh, W.; Maginnis, F.; Brown, W.: Enterprise Application Integration: A Wiley Tech Brief. John Wiley & Sons, 2000.

[Sa02] Satyanarayanan, M.: The Evolution of CODA. In: ACM Transactions on Computer Systems 20 (2), 2002, S.85-124.

[SK92] Sheth, A. P.; Kashyap, V.: So Far (Schematically) yet So Near (Semantically). In:Database Semantics Conference on Interoperable Database Systems, Lorne, Victoria, Australia, 1992 S.283-312.

[SS00] Saake, G.; Sattler, K.-U.: Datenbanken und Java. JDBC, SQLJ und ODMG. dpunkt-Verlag, 2000.

[SSN02] Sharma, R.; Stearns, B.; Ng, T.: J2EE Connector Architecture and Enterprise Application Integration. Addison Wesley, 2002.

[TG82] Traiger, I. L.; Gray, J.; Galthier, C. A. et al.: Transactions and consistency in distributed database systems. In: ACM Transactions on Database Systems 7 (3), 1982, S.323-342.

[TTP95] Terry, D. B.; Theimer, M. M.; Petersen, K. et al.: Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In:Symposium on Operating Systems Principles, Copper Mountain Resort, Colorado, 1995 S.172-183.

Nutzerdefinierte Replikation zur Realisierung neuer mobiler Datenbankanwendungen

Christoph Gollmick

Friedrich-Schiller-Universität Jena, Institut für Informatik
Ernst-Abbe-Platz 2, 07743 Jena
gollmick@informatik.uni-jena.de

Kurzfassung: Replikation ist für die unverbundene Verfügbarkeit von Daten in mobilen Umgebungen unverzichtbar. Das Konzept der nutzerdefinierten Replikation erweitert die Möglichkeiten zum Umgang mit replizierten relationalen Datenbankinhalten um die anwendungsgesteuerte dynamische Auswahl von Replikaten. Die nutzerdefinierte Replikation bietet dazu sowohl dem Administrator als auch dem Anwendungsprogrammierer, an SQL angelehnte, deskriptive Schnittstellen. Über diese Schnittstellen werden die zur Replikation bereitgestellten Daten, die Datenauswahl durch die mobile Anwendung sowie die jeweils damit verbundenen Zusicherungen und Konfliktbehandlungsmöglichkeiten spezifiziert. Der Beitrag stellt die neue Funktionalität vor und erläutert die Verwendung der Schnittstellen am Beispiel des interaktiven Reiseinformationssystems HERMES.

1 Einführung und Motivation

Die Computertechnologie wurde in den letzten Jahren um eine Dimension erweitert, die Mobilität von Geräten und Nutzern. Mobile Geräte sind dabei abhängig von unterwegs verfügbaren Kommunikationsmitteln und einer mobilen Energieversorgung. Beide Voraussetzungen sind heute und auch in naher Zukunft noch mit erheblichen Einschränkungen versehen. Der Stand der Technik bietet zwar adäquate drahtlose Kommunikationsmittel wie WLAN oder UMTS, diese sind aber noch nicht weit verbreitet, teuer und energieintensiv in der Benutzung. Außerdem sind sie prinzipbedingt nicht so leistungsstark und sicher gegen Angriffe von Dritten wie feste Netze. Das Problem kurzer Akkulaufzeiten, das durch (drahtlose) Kommunikation verstärkt wird, ist ebenfalls noch ungelöst. Für den jederzeitigen, sicheren und kostengünstigen Zugriff lokaler Anwendungen auf Datenbankinhalte (mobiler Datenbankzugriff) sind deshalb die Möglichkeit zum *zeitweise unverbundenen Arbeiten* mobiler Clients [Gol01, JHE99] und geeignete *Replikationstechniken* [ÖV99, HHB96] erforderlich.

Heutige kommerzielle mobile Datenbanklösungen [Fan00] erlauben sowohl eine isolierte Arbeit mobiler Clients als auch die Integration in eine kooperative Datenverwaltung mit Workstation- und Großrechner-DBS. Ihre Fähigkeiten sind dabei auf die mobile Umsetzung traditioneller Anwendungen (z. B. Zugriff auf ein SAP R/3-System oder die Datenbank für einen Versicherungsvertreter) zugeschnitten. Bei diesen Anwendungen sind die

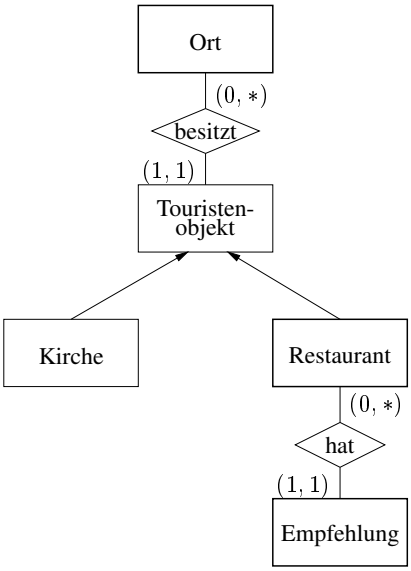
Tätigkeit eines jeden Nutzers und damit die zu replizierenden Daten bekannt und es besteht wenig Konfliktpotential. Die Definition von Replikationsabbildungen, die Datenauswahl für mobile Clients und die Nutzerverwaltung können durch den Administrator auf der Serverdatenbank bzw. einem zwischengeschalteten Synchronisationsserver erfolgen (*zentralistischer Ansatz*). Die Fähigkeiten zur automatischen Behandlung von Änderungskonflikten beschränken sich auf einfache, attributorientierte, Strategien. In der Praxis sind deshalb bei komplexeren Anwendungen eigene anwendungsabhängige Konfliktbehandlungsprozeduren oder sogar ein manueller Eingriff erforderlich.

Die Mobilität von Nutzern bietet über die Bereitstellung klassischer Dienste hinaus aber auch Chancen für die Entwicklung ganz neuer datenbankgestützter mobiler Anwendungen. Vorstellbar sind Anwendungen, die dem mobilen Nutzer zum einen auf seinen Aufenthaltsort und seine aktuelle Tätigkeit zugeschnittene Daten lokal zur Verfügung stellen und zum anderen die Interaktion zwischen Nutzern und dem System ermöglichen. Das heißt, Nutzer sollen unmittelbar Vorort neue Daten und Änderungen an gemeinsam genutzten Daten ins System einbringen können. Beispiele solcher Anwendungen sind Unterstützungssysteme für Katastrophenhilfskräfte oder das, im Rahmen dieses Projektes entwickelte, interaktive mobile Reiseinformationssystem HERMES [Bau03]. Diese neuen Anwendungsszenarien stellen andere Anforderungen an die Datenhaltungskomponente eines Informationssystems als klassische Anwendungen. Dazu gehört ein Dienst, über den eine mobile Anwendung Daten dynamisch zur Laufzeit, abhängig von der Zeit, dem Ort und der aktuellen Tätigkeit ihrer Nutzer für unverbundene Zeiten zur Replikation auswählen kann – *nutzerdefinierte Replikation* (NR). Wie ein solcher Dienst gestaltet werden kann und welche damit zusammenhängenden Probleme gelöst werden müssen, zeigt dieser Beitrag.

Der Beitrag ist wie folgt gegliedert. Abschnitt 2 stellt das mobile Reiseinformationssystem HERMES und den zur Illustration verwendeten Datenbankausschnitt vor. In Abschnitt 3 werden die charakteristischen Eigenschaften und das Arbeitsmodell der nutzerdefinierter Replikation anhand eines Beispiels erläutert. Für die Realisierung der NR ist u. a. eine effiziente Verwaltung dynamisch replizierter Nutzerdaten erforderlich, Abschnitt 4 gibt einen Überblick zur prototypischen Umsetzung. Abschnitt 5 schließt mit einer Zusammenfassung und einem Ausblick auf noch zu lösende Probleme und mögliche Erweiterungen.

2 Das Reiseinformationssystem HERMES

Das datenbankbasierte Reiseinformationssystem HERMES hat die Aufgabe, seine Nutzer bei der Reiseplanung zu unterstützen und ihnen unterwegs ortsabhängige Informationen bereitzustellen. Abrufbar sind beispielsweise Informationen über Orte, Sehenswürdigkeiten, historische Gebäude, Hotels, Restaurants oder auch empfehlenswerte Routen. Im Unterschied zu klassischen Touristenführern werden aber nur die Basisdaten, wie Name, Einwohnerzahl oder Historie von Orten, zentral bereitgestellt. Der weitaus größte Teil wird von den mobilen Nutzern selbst und in der Regel vor Ort ergänzt und aktualisiert. Hierzu zählen beispielsweise Reiseberichte/Bewertungen, besuchte Routen, Öffnungszeiten von Gebäuden, Preise und Erfahrungen mit Hotels oder Speisekarten von Restaurants. HERMES ist also ein mobiles Reiseinformationssystem von Nutzern für Nutzer.



(a) E/R-Modell

Ort	Name	Land	Einwohner
	München	Bayern	1300000
	Erfurt	Thüringen	220000
	Jena	Thüringen	100000

Kirche	Name	Ort	e/k
	Friedenskirche	Jena	e
	Stadtkirche	Jena	e
	Peterskirche	München	k

Restaurant	Nr	Name	Ort
	1	Zur Noll	Jena
	2	Roter Hirsch	Jena
	3	Hohe Warte	Erfurt
	4	Brotzeitstüberl	München

Empfehlung	Von	RNr	Text
	Erwin	1	„gut“
	Egon	1	„na ja“
	Egon	2	„geht so“

(b) Eine relationale Umsetzung

Abbildung 1: Beispiel

Die angestrebte Funktionalität und die Verwendung von HERMES soll an einem einfachen Beispiel (Abbildung 1, *Restaurant.Ort* und *Kirche.Ort* sind jeweils Fremdschlüssel zur Tabelle *Ort*, *Empfehlung.RNr* ist Fremdschlüssel bzgl. *Restaurant* und es gilt `UNIQUE`(*Restaurant.Name*, *Restaurant.Ort*)) skizziert werden. Nach der Ankunft in Jena möchte sich ein Reisender über die hiesigen Kirchen informieren und wählt über die Oberfläche der lokalen HERMES-Anwendung auf seinem PDA die gewünschten Informationen aus, die nach einer Verbindungsaufnahme vom zentralen Server auf sein Gerät übertragen werden. Während des Stadtbummels kann er auf die replizierten Informationen offline zugreifen. Nach dem Stadtbummel möchte er sich zum Abendessen ein gutes Restaurant wählen. Dazu selektiert er die Daten der Restaurants mit den Nutzerempfehlungen. Nach dem ausführlichen Studium der Informationen entscheidet er sich für das Restaurant „Zur Noll“. Auf dem Weg dorthin kommt er am Restaurant „Ratszeise“ vorbei und entscheidet sich spontan für diesen Gastronomiebetrieb. Da die „Ratszeise“ noch nicht im Restaurantkatalog verzeichnet ist, fügt er ihre Daten ein. Am nächsten Morgen synchronisiert er seine Änderungen mit der zentralen Datenbank.

Dieses kleine Beispiel beinhaltet bereits die meisten der zu lösenden Fragestellungen und Probleme, die wir mit dem Konzept der nutzerdefinierten Replikation adressieren wollen. Die Anwendung muß die Möglichkeit haben, nach den Vorgaben des Nutzers, geeignete Datenmengen vom Server zur Laufzeit zu replizieren. Bereits lokal vorhandene und noch aktuelle Daten sollen dann nicht erneut repliziert werden. Für den Fall, daß der lokale

Speicherplatz nicht mehr ausreicht, müssen replizierte Daten wieder gelöscht werden. Für das Hinzufügen von Restaurants muß dem mobilen Nutzer mindestens das Einfügerecht auf der Tabelle *Restaurant* gewährt werden und eine automatische Konfliktbehandlung muß für die Konsistenz der Daten bei der Synchronisation sorgen.

	traditionelle Anwendungen	HERMES
Nutzeranzahl	begrenzt, stabil	potentiell unbegrenzt, variabel
Datenauswahl		
◦ Wo?	zentral	durch Nutzer
◦ aufgabenabhängig	ja	ja
◦ ortsabhängig	selten	häufig
◦ Variabilität	gering	hoch
◦ Änderungskonflikte	selten	häufig

Tabelle 1: Vergleich traditioneller mobiler Anwendungen mit HERMES

Tabelle 1 stellt die charakteristischen Eigenschaften von HERMES denen der traditionellen Anwendungen gegenüber. HERMES steht dabei stellvertretend für eine Klasse von neuen mobilen Datenbankanwendungen, die sich durch die Forderung nach einer dynamischen, orts-, aufgaben- und zeitabhängigen Datenauswahl auszeichnen.

3 Die nutzerdefinierte Replikation

Trotz ihrer Fokussierung auf den zentralistischen Ansatz bieten einige kommerzielle mobile Datenbanklösungen (z. B. Oracle 9i lite) der mobilen Anwendung die Möglichkeit, über die Definition materialisierter Sichten aus einer vorher für die Replikation vorbereiteten Datenmenge auszuwählen. Für die Unterstützung einer großen Zahl von Nutzern mit dynamischer Datenauswahl fehlen jedoch wichtige Elemente, wie eine einfach zu handhabende Konfliktbehandlung und eine effiziente Verwaltung replizierter Daten sowohl auf dem Client als auch auf dem Server. Mit der Konzeption der nutzerdefinierten Replikation für relationale Datenbankinhalte verbinden wir die folgenden Ziele:

- Bereitstellung einer Schnittstelle für mobile Anwendungen, die es erlaubt, zur Laufzeit Daten aufgaben-, orts-, und zeitabhängig in Verbindung mit Zusicherungen (z. B. konfliktfreie Änderbarkeit) zur Replikation anzufordern.
- Wo möglich, Bereitstellung deskriptiver, an SQL angelehnter, Konstrukte sowohl für den Administrator als auch für den Anwendungsprogrammierer. Dieses Ziel soll insbesondere für die Spezifikation der Konfliktbehandlung verfolgt werden.
- Ein weiteres wichtiges Ziel ist die Trennung der Aspekte der Replikatdefinition (Auswahl zu replizierender Daten durch die Anwendung) von den Aspekten der Replikatverwaltung (Anlegen von Metadaten für die Synchronisation, Caching/Prefetching-Verfahren, etc.), die in kommerziellen Systemen oft eng verknüpft sind. Auf die Re-

plikatverwaltung und eine mögliche Realisierung der NR werden wir in Abschnitt 4 kurz eingehen.

In einer mobilen Replikationsumgebung gibt es zwei grundlegende Aktivitäten, die *Replikationsdefinition* und die *Synchronisation*. Beide Aktivitäten sind in das *Arbeitsmodell* eingebettet, das Voraussetzungen und Abläufe vorgibt.

3.1 Replikationsdefinition

Bei der Replikationsdefinition wird eine Auswahl getroffen, welche Daten eines Quellsystems (*Replikationsquelle*) auf dem Zielsystem (*Replikationsziel*) repliziert verfügbar sein sollen (*Replikate*), für wen welche Operationen auf den Replikaten erlaubt sind (*Zusicherungen*) und welche Konfliktvermeidungs-, -erkennungs-, und -auflösungsstrategien (*Konfliktbehandlung*) für die Replikate Verwendung finden. Als Replikationsquelle sind bei der NR nur stationäre Server erlaubt, Replikationsziele sind die mobilen Clients.[1] Die Replikationsdefinition gliedert sich in die zwei Unteraktivitäten *Replikationsschemadefinition* und *Replikatdefinition*:

1. Replikationsschemadefinition

   Das *Replikationsschema* ist der Ausschnitt des Schemas einer relationalen Datenbank auf der Replikationsquelle, der für die Replikation auf mobile Clients sichtbar ist, erweitert um Zusatzinformationen. Zum Replikationsschema gehören Zusicherungen für mögliche Operationen (z. B. lokal änderbar) sowie alle zur Synchronisation notwendigen Erweiterungen, sofern sie sich auf Elemente der Schemadefinition (z. B. Datentypen, Integritätsbedingungen) beziehen.[2] Die *Replikationsschemadefinition* ist bei der NR, wie beim zentralistischen Ansatz, Aufgabe eines Administrators.

2. Replikatdefinition

   Bei der *Replikatdefinition* werden, bezogen auf ein Replikationsschema, die für eine Aufgabe, einen Zeitpunkt und einen Ort benötigten Daten ausgewählt und die bei der Replikationsschemadefinition festgelegten Zusicherungen und Konfliktbehandlungsmethoden instantiiert. Die Definition der Replikate wird bei der NR, meist auf Wunsch des Nutzers, durch die mobile Anwendung durchgeführt. Ein Client kann beliebig viele Replikate anlegen lassen, geforderte Zusicherungen in gewissem Rahmen ändern und nicht mehr benötigte Replikate löschen.

3.2 Synchronisation

Da es über die Replikation mehrere Kopien eines Datums bei verschiedenen mobilen Clients geben kann, auf denen unabhängig voneinander lokal gearbeitet wird, ist eine *Synchronisati-*

[1] In Anbetracht der exponierten Stellung mobiler Clients und der angedachten Nutzerzielgruppe verbieten wir, daß Primärkopien gemeinsam genutzter Daten auf mobilen Clients liegen.

[2] Das Anlegen eines *Key Pool* zur Einfügekonfliktvermeidung für einen Schlüsselkandidaten gehört zum Replikationsschema. Die Bereitstellung einer konkreten Anzahl von Schlüsseln aus dem Pool gehört zur *Replikatdefinition*.

on untereinander und mit der Replikationsquelle erforderlich. Ziel ist dabei die Aufrechterhaltung oder Wiederherstellung eines, für alle Beteiligten, konsistenten Datenbankzustands. Bei der nutzerdefinierten Replikation erfolgt die Synchronisation immer zwischen einem Client und dem Server.[3] Die Synchronisation von Client und Replikationsquelle besteht aus zwei Phasen, der *Reintegration* und der *Rückübertragung*:

1. Reintegration

   Bei der *Reintegration* werden lokale Änderungen an Replikaten beim Server eingebracht und der nachgelagerten Integritätsprüfung sowie einer Konfliktbehandlung (primärschlüsselorientierte Konflikterkennung) unterzogen. Die Synchronisation der NR orientiert sich dabei an der in [GHOS96] vorgestellten transaktionsorientierten *Two-Tier-Replication*. Dabei werden unverbunden lokal ausgeführte Änderungstransaktionen auf der Replikationsquelle mit aktuellen Daten erneut ausgeführt, wo sie ihr endgültiges Commit oder Abort erfahren.

2. Rückübertragung

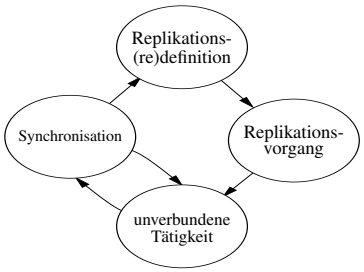
   Bei der *Rückübertragung* vom Server wird ein aktueller transaktionskonsistenter, u. U. konfliktbereinigter, Zustand der zur Replikatdefinition gehörenden Daten an den Client übermittelt.

Es ist zu beachten, daß für den kompletten Reintegrationsprozeß eines mobilen Client die Transaktionseigenschaften *Konsistenz* (für den Zustand der Replikationsquelle) und *Isolation* erfüllt sein müssen.[4] Dies kann beispielsweise durch eine Serialisierung der Reintegrationsprozesse mehrerer Clients erreicht werden.

3.3 Arbeitsmodell

Wir vertreten die Meinung, daß die Akzeptanz von HERMES entscheidend von zwei Punkten abhängt, der Verfügbarkeit der vollen Funktionalität am „Ort der Information“ und geringen (Übertragungs-)Kosten. Die Möglichkeit unverbundener Tätigkeit hat deshalb in unseren Betrachtungen einen hohen Stellenwert. Das Arbeitsmodell der aktuellen Konzeptspezifikation unterscheidet nur zwischen *verbundenem* und *unverbundenem* Zustand eines mobilen Client (siehe Abschnitt 5). Die Voraussetzung für unverbundenes Arbeiten ist ein ausreichend dimensionierter Client mit einem lokalen DBS, das transaktionsorientierte Synchronisation beherrscht (siehe Produktübersicht in [Fan00]).

Abbildung 2: Ablauf der NR



Die Beschreibung des Ablaufs bei der nutzerdefinierten Replikation (Abbildung 2) soll anhand einer Umsetzung des zweiten Teils des Beispielszenarios aus Abschnitt 2 (Restaurantbesuch) erfolgen. Für eine Beschreibung von Syntax und Semantik der hier vereinfacht dargestellten Konstrukte sei auf [Mül03] verwiesen.

[3] Der Abgleich zwischen gleichberechtigten Clients schafft eine Reihe zusätzlicher (Konsistenz-)Probleme (z. B. weiß nicht jeder Client über die Replikatdefinition eines anderen Bescheid), die wir in der aktuellen Konzeptspezifikation vermeiden wollen.

[4] Atomarität (und Dauerhaftigkeit für lokale Transaktionen) kann nicht gewährleistet werden, da die Reintegration einzelne lokale Transaktionen zurückweisen kann.

1. Der Administrator gibt die Tabellen *Restaurant* und *Empfehlung* der Server-Datenbank zur Replikation frei in dem er *konsolidierte Tabellen* (siehe Abschnitt 4), als Elemente des Replikationsschemas, anlegt:

```
CREATE CONSOLIDATED TABLE ConsRestaurant
SOURCE Restaurant
FOR OFFLINE INSERT ADD KEYPOOL (Nr) DEFAULT 1 MAX 5
FOR OFFLINE DELETE
DEFAULT RESOLUTION DISCARD;

CREATE CONSOLIDATED TABLE ConsEmpfehlung
SOURCE Empfehlung
FOR OFFLINE INSERT FOR OFFLINE DELETE
FOR OFFLINE UPDATE (Text)
DEFAULT RESOLUTION DISCARD;
```

   In beide konsolidierten Tabellen kann eingefügt werden, für den Primärschlüssel der Tabelle *ConsRestaurant* wurde vom Administrator die Konfliktvermeidungsstrategie `KEYPOOL` ausgewählt. Die mobile Anwendung kann maximal 5 Schlüssel gleichzeitig reservieren, Default-Wert ist 1 Schlüssel.[5] Als Konfliktauflösungsstrategie wurde festgelegt, die vorläufigen Änderung des *Client* zu verwerfen.

2. Die Anwendung des Reisenden fordert im *verbundenen* Zustand von der Replikationsquelle die Daten der Jenaer Restaurants zur Replikation an:

```
CREATE REPLICATION VIEW RepRestaurantJena
AS SELECT Nr, Name, Ort FROM ConsRestaurant
   WHERE Ort='Jena'
FOR OFFLINE MODIFICATION WITH CHECK OPTION;

CREATE REPLICATION VIEW RepEmpfehlungJena
AS SELECT Von, RNr, Text FROM ConsEmpfehlung
   WHERE RNr IN (SELECT Nr FROM ConsRestaurant
                 WHERE Ort='Jena')
FOR OFFLINE MODIFICATION WITH CHECK OPTION;

SYNCHRONIZE ALL;
```

   Die zu replizierenden Daten werden über den `SELECT`-Teil der `CREATE REPLICATION VIEW`-Anweisung bestimmt. Standardmäßig ist eine `REPLICATION VIEW` (RV) nicht änderbar, werden aber Angaben zu `FOR OFFLINE` gemacht, so wird sie, wenn die Rückabbildung möglich ist, für unverbundene Modifikationen bereitgestellt. Optional können Parameter für die Konfliktbehandlung angegeben werden.

   Nach dem erfolgreichen `SYNCHRONIZE ALL` steht der mobilen Anwendung eine Sicht zur Verfügung, welche die in der RV-Definition selektierten und replizierten

[5] Alternativ könnte der nutzerdefinierte Replikationsdienst die Anzahl benötigter Schlüssel für einen Client anhand der Häufigkeit der Einfügungen zwischen zwei Synchronisationsintervallen in der Vergangenheit bestimmen.

Daten referenziert. Auf eine solche Sicht können lokale Anfragen, Sicht- und Cursor-Definitionen aufsetzen. Die angegebenen Beispielkonstrukte erzeugen auf dem mobilen Client die zwei unverbunden änderbaren RVs *RepRestaurantJena* und *RepEmpfehlungJena*.

Wichtig ist, daß eine RV-Definition vollständig unabhängig von den tatsächlich replizierten Daten ist. Die Replikationsquelle entscheidet bei der Auswertung von `CREATE REPLICATION VIEW`, welche Daten für die geforderten Zusicherungen und unter Berücksichtigung der schon lokal vorhandenen Daten auf den Client repliziert werden müssen. Eine mobile Anwendung kann damit ohne redundante Datenhaltung dynamisch beliebige RVs, auch mit überlappender Definition, anlegen und wieder löschen. Das Löschen einer RV (im verbundenen Zustand) signalisiert dem System, daß die von der Sicht referenzierten Daten nicht mehr benötigt werden. Die betroffenen Replikate und ihre Metadaten werden, sofern sie von keiner anderen RV mehr benötigt werden, daraufhin aufgelöst.

3. Im *unverbundenen* Zustand fügt der Nutzer über seine HERMES-Anwendung einen neuen Datensatz in die RV *RepRestaurantJena* der lokalen Datenbank ein:

   ```
   INSERT INTO RepRestaurantJena
   VALUES (Nr_VALUE(), 'Ratszeise', 'Jena');
   ```

   `Nr_VALUE()` stellt dabei für den Primärschlüssel der Quelltabelle einen Schlüsselwert aus dem lokalen *Key Pool* bereit.[6]

   Bei der RV-Definition haben wir auch eine änderbare Kopie der *ConsEmpfehlung*-Tabelle angefordert. Wir könnten also beispielsweise die Empfehlungen eines anderen Nutzers aus der Datenbank löschen. Das zeigt, daß die Realisierung von Zusicherungen eng an die Verfügbarkeit einer inhaltsbasierten Authentisierung und Rechteverwaltung gekoppelt ist, wie sie von kommerziellen DBMS nicht unterstützt bzw. der jeweiligen Anwendung überlassen wird. Der Entwurf der nutzerdefinierten Replikation sieht ein solches Rechtekonzept vor [Mül03], wir werden hier jedoch nicht darauf eingehen.

4. Mit der anschließenden Synchronisation kann, bei positivem Ausgang der Konfliktauflösung, die Dauerhaftigkeit der lokalen Änderungen erzielt werden. Sollte ein anderer Restaurantbesucher bereits früher einen Eintrag zur „Ratszeise“ reintegriert haben, wird die eigene Änderung verworfen.

4 Realisierung der NR

Die prototypische Realisierung der nutzerdefinierten Replikation wird zur Zeit im Rahmen mehrerer Arbeiten betrieben (Architektur [Mül03], Konfliktbehandlung [Lie03] und Replikatspeicherung [Gol02a]). Über die am Anfang von Abschnitt 3 formulierten allgemeinen Ziele hinaus fordern wir von einer Umsetzung: Skalierbarkeit in der Anzahl der mobilen Clients, Reduktion von Verbindungszeiten und Flexibilität bzgl. der Geräte/Produkte für

[6] Für eine weitere Einfügung müßte der *Key Pool* im verbundenen Zustand zunächst wieder aufgefüllt werden.

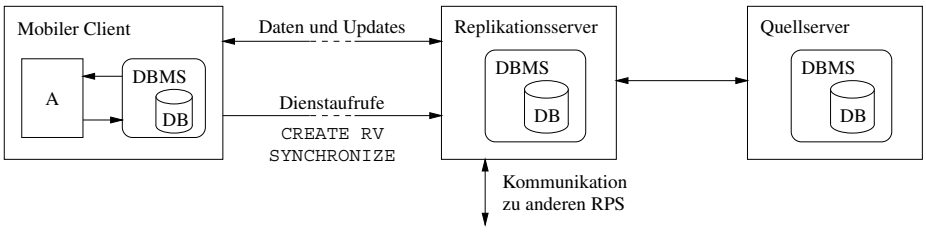


Abbildung 3: RPS-Architektur zur Realisierung der NR

Replikationsquellen und -ziele. Diese Anforderungen gelten zwar auch für die zentral administrierte Replikation, sind aber bei der NR besonders problematisch, da Anzahl und Verhalten der mobilen Nutzer nicht im voraus detailliert bekannt sind.

Die von uns favorisierte 3-stufige Architektur [Gol02b] ist in Abbildung 3 dargestellt. Die nutzerdefinierte Replikation der Server-Daten wird dabei von einem oder mehreren zwischengeschalteten Replikationsservern (RPS – *Replication Proxy Server*) vermittelt,[7] die später auch geographisch verteilt sein sollen. Im Unterschied zu kommerziell erhältlichen Synchronisationsprodukten, hält ein RPS über stationäre asynchrone Replikation eine Kopie (konsolidierte Tabellen) der Quellserver-Daten. Damit wird eine Entkopplung der Arbeit mobiler Nutzer vom Quellserver erreicht. Dies erhöht Verfügbarkeit und Sicherheit und gibt uns die Möglichkeit, die replizierten Daten auf dem RPS für die erfaßten Zugriffs- und Mobilitätsprofile der Clients optimal zu (re-)organisieren.

Die zwei wichtigsten Aufgaben eines RPS sind die automatische Anlage von Daten- und Schemaelementen für die Änderungserfassung und die Konfliktbehandlung (z. B. die Auslesefunktion `Nr_VALUE`) sowie die Bestimmung der Daten und Schemaelemente die, unter Berücksichtigung der bereits lokal vorhandenen, tatsächlich übertragen werden müssen. Der zweite Punkt wirft zwei Fragen auf, wie kann ich Daten semantisch gruppieren und wie kann bei gegebener Anfrage eine minimale Menge zu replizierender Granulate bestimmt werden. Diese Fragen haben wir in [Gol02a] untersucht und ein Fragmentkonzept für relationale Daten entwickelt.

5 Zusammenfassung und Ausblick

In Abgrenzung zu traditionellen Anwendungen haben wir eine Klasse neuer mobiler Datenbankanwendungen untersucht, die eine dynamische, orts-, aufgaben- und zeitabhängige Replikation von Daten erfordern. Zur Unterstützung dieser Anwendungsszenarien wurde das Konzept der nutzerdefinierten Replikation für relationale Datenbankinhalte entwickelt, dessen Funktionalität und Schnittstellen am Beispiel der Anwendung HERMES erläutert wurden.

Die prototypische Umsetzung der nutzerdefinierten Replikation ist im Rahmen mehrerer Arbeiten eingeleitet. Dennoch müssen eine Reihe von Teilproblemen, besonders im Bereich der automatischen Konfliktbehandlung, noch konzeptuell gelöst werden. So sind,

[7] Die Replikationsquelle für die mobilen Clients ist der RPS, nicht der eigentliche Quellserver.

für den Fall einer erfolglosen automatischen Konfliktauflösung, der Anwendung geeignete Informationen für eine Einbeziehung des Nutzers bereitzustellen (z. B. Handynummer des Konfliktpartners). Als Arbeitsmodell haben wir zunächst nur den wichtigen Fall der vollständig unverbundenen Arbeit betrachtet. In der Praxis finden sich jedoch auch Kommunikationsmittel, die für eine Datenübertragung ungeeignet sind, aber die Übermittlung von Status-Informationen erlauben (z. B. SMS zur Übermittlung von Sperrfreigaben). Die damit verbundenen Möglichkeiten sind zu analysieren und, in einem zweiten Schritt, geeignet in das Konzept zu integrieren.

Literaturverzeichnis

[Bau03] K. Baumgarten. Konzept und Realisierung des interaktiven Reiseinformationssystems HERMES. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2003. In Vorbereitung.

[Fan00] T. Fanghänel. Vergleich und Bewertung kommerzieller mobiler Datenbanksysteme. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, September 2000.

[GHOS96] J. Gray, P. Helland, P. O'Neil und D. Shasha. The Dangers of Replication and a Solution. In *Proc. of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Canada*, Seiten 173–182. ACM, 1996.

[Gol01] C. Gollmick. Unterstützung mobiler Clients durch erweiterte Client/Server-Datenbanksysteme. In *Tagungsband zum 13. GI-Workshop „Grundlagen von Datenbanken", Gommern, Deutschland*, Preprint Nr. 10, Seiten 43–47. Fakultät für Informatik, Universität Magdeburg, Juni 2001.

[Gol02a] C. Gollmick. Konzept und Anforderungen der nutzerdefinierten Replikation zur Realisierung neuer mobiler Datenbankanwendungen. Jenaer Schriften zur Mathematik und Informatik Math/Inf/15/02, Institut für Informatik, Friedrich-Schiller-Universität Jena, September 2002.

[Gol02b] C. Gollmick. Using Replication Proxy Servers for Scalable Mobile Database Access. In *Proc. of the EDBT 2002 PhD Workshop, Prag, Tschechische Rep.*, Seiten 115–118. MATFYZPRESS, März 2002.

[HHB96] A. Helal, A. Heddaya und B. Bhargava. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, August 1996.

[JHE99] J. Jing, A. Helal und A. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2):117–157, 1999.

[Lie03] M. Liebisch. Synchronisationskonflikte beim mobilen Datenbankzugriff: Vermeidung, Erkennung, Behandlung. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2003. In Vorbereitung.

[Mül03] T. Müller. Architektur und Prototyp eines Replication Proxy Server für die nutzerdefinierte Replikation von Datenbankinhalten. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2003. In Vorbereitung.

[ÖV99] T. Özsu und P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, Inc., 2. Auflage, 1999.

Rechnergestützte Suche nach Korrelationen in komplexen Datensätzen der Biowissenschaften

Stephan Heymann, Katja Tham, Peter Rieger and Johann-Christoph Freytag

Lehrstuhl für Datenbanken und Informationssysteme,
Institut für Informatik, Humboldt Universität zu Berlin,
Unter den Linden 6, D-10099 Berlin.
E-mail: {heymann, tham, rieger, freytag}@dbis.informatik.hu-berlin.de

Abstract: Umfangreiche Datensammlungen, wie sie im Ergebnis molekularbiologischer Hochdurchsatzexperimente entstehen, sind zu einer wichtigen Triebfeder der funktionalen Genomforschung geworden. Eine wesentliche Anforderung an die Informationstechnologie besteht darin, Werkzeuge bereitzustellen, die es ermöglichen, implizite Zusammenhänge in Datensätzen unterschiedlicher methodischer Herkunft aufzudecken. Diese Arbeit beschreibt eine Vorgehensweise, die von der Aufbereitung der Daten bis zur Visualisierung der Ergebnisse einen integrierten Lösungsansatz bereitstellt. Im ersten Schritt werden die Experimentaldaten so genannten Datenkategorien zugeordnet. Entsprechend dieser Systematik erfolgt die Integration in ein relationales Datenbank-Managementsystem. In einer Aufbereitungsphase werden die Daten nach den Anforderungen des Anwenders in ein komplexes Analysemodell überführt. Dieses dient dem Auffinden unbekannter Zusammenhänge, als Eingangsformat für die grafische Visualisierung sowie der interaktiven Navigation. Im Rahmen dieser Vorgehensweise wurde ein Verfahren entwickelt, das es ermöglicht, qualitative Aussagen über Korrelationen in Daten unterschiedlicher Kategorien abzuleiten.

1 Einleitung

Die Erkennung von Erklärungsmustern für komplexe Lebensvorgänge aus Genomdaten ist eines der zentralen Anliegen der Bioinformatik. Es existiert jedoch keine einheitliche "Genomtheorie", die erklärte, *wie* das Erbgut eine Spezies und einzelne Individuen befähigt zu leben, zu agieren, zu reagieren, sich anzupassen, zu entwickeln oder auszusterben.

Historisch gewachsenes Domänenwissen
Die Facetten des komplexen funktionellen Wechselspiels der Gene und ihre wechselseitigen Abhängigkeiten werden im Ergebnis eines ganzen Arsenals adäquater Experimentaltechniken und computergestützter Untersuchungen mosaikartig zusammengetragen. Die Erkenntnisse bleiben jedoch in den Grenzen der Aussagefähigkeit der jeweiligen Methode gefangen. Demzufolge hängt der Erkenntniszuwachs in hohem Maße von Technologien ab, die einen Forscher beim Aufdecken impliziter Korrelationen zwischen Datensätzen unterschiedlichen methodischen Ursprungs unterstützen. Einen besonderen Schwerpunkt bildet dabei die Auswertung von Messreihen aus Hochdurchsatzverfahren, deren Querbezüge dann offenbar werden, wenn ein experimenteller Datensatz zu einem zweiten – oder zu mehreren – in Bezug gesetzt und ausgewertet wird.

Der gemeinsame Kern heterogener Experimentaldatensätze
Abstrakt betrachtet baut die Analyse von Experimentaldatensätzen auf folgenden Überlegungen auf: Gegeben sei ein Sachverhalt, der n Objekte O einer Klasse überspannt. Die Menge $M = \{O_i\}$, i=1, ...,n werde k methodisch unterschiedlichen Untersuchungen unterzogen, wodurch jeweils Aussagen zu Untermengen $M^{(k)} \subseteq M$ gewonnen werden. Untermengen deswegen, weil im Regelfalle nicht alle Elemente aus M in die Untersuchungen einbezogen sind oder aber die Untersuchungsmethode nicht zu allen untersuchten Elementen Ergebnisse liefert. Der Charakter der Untersuchungen wiederum bedingt, dass die Resultate in drei Datenkategorien fallen:

(i) Eigenschaften und Merkmale, die *einzelnen* Objekten der Teilmenge $M^{(k)}$ als Deskriptoren zugeordnet werden können. Ein Objekt hat die betreffende(n) Eigenschaft(en) oder es hat sie nicht.

(ii) Beziehungen einer bestimmten Qualität zwischen je zwei Objekten einer Teilmenge $M^{(k)}$. Die Vereinigungsmenge aller *paarweisen* Beziehungen einer Qualität lässt sich am besten durch ein Netzwerk darstellen. Es ist unerheblich, ob die paarweisen Beziehungen in ihrer Gesamtheit ein zusammenhängendes Netzwerk bilden oder nicht.

(iii) Beziehungen und/oder Attribute, die einer Gruppe von Objekten – nicht einzelnen Gruppenmitgliedern – innerhalb einer Teilmenge $M^{(k)}$ eigen sind und *mengenwertige* Unterscheidungs- und Zuordnungskriterien darstellen.

Auf dem Gebiet der funktionellen Genomanalyse liegt es nahe, die Gene/Genprodukte als Objekte zu behandeln und M als die Vereinigungsmenge der Gene einer Spezies anzusetzen. Bei der Bäckerhefe *Saccharomyces cerevisiae* beispielsweise beinhalten ~6.300 Gene die Synthesevorschrift für die entsprechenden Proteine. Vom Standpunkt moderner Datenhaltung aus gesehen ist das ein leicht zu handhabender Basisdatensatz geringen Umfangs, der sich seit dem Abschluss des Hefe-Sequenzierungsprojekts 1997 (siehe z.B. [Me97]) nur noch marginal veränderte. Demgegenüber führt die weltweite Erforschung der Funktionalität dieser Objekte auch weiterhin zu erheblichem Datenaufkommen in allen drei oben genannte Kategorien, die zum Verständnis der Lebensvorgänge im Modell- und Produktionsorganismus Hefe beitragen.

Hefegenomik – Sachstand, Trends und Erfordernisse aus Informatiksicht
Mit dem Aufkommen experimenteller Hochdurchsatztechniken und im Ergebnis systematischer Studien lässt sich beobachten, dass sich der Erkenntniszuwachs von Kategorie (i) mehr und mehr in die Kategorie (iii) verschiebt. Folglich erscheint es von besonderer Wichtigkeit, solche Herangehensweisen zu forcieren, die einen ganzheitlichen Blick auf die ständig wachsenden Daten erlauben, indem die o. g. Kategorien kombiniert und zueinander in Beziehung gesetzt werden.

Die im Rahmen der Experimente gesammelten Daten, werden heute in der Regel in Form von Dokumenten im Internet bereitgestellt. Um Zusammenhänge in den Daten aufzudecken, ist es jedoch erforderlich, dass spezifische Sichten auf diese heterogene Datenbestände definiert werden können. Diese Notwendigkeit erfordert eine Integration, die über die Inhalte einzelner Datenquellen hinausgeht. Jedoch ist im Bereich der Le-

benswissenschaften die Integration häufig auf die Bereitstellung von Dokument-Dokument-Querbezügen mit Hilfe von Hypertext-Markups beschränkt. Das ist jedoch für die Anforderungen einer integrierten Analyse unzureichend, da in der Regel weder die exakte Semantik eines Querbezugs definiert ist[1], noch können dokumentübergreifende Anfragen oder mengenwertige Ergebnisse verarbeitet werden.

Für die Hefedaten (siehe Abschnitt 3) sind Dienste verfügbar, die den Zugriff auf mehrere Datenquellen unter einer gemeinsamen Nutzerschnittstelle ermöglichen. Es werden dabei jedoch nur solche Zusammenhänge berücksichtigt, die bereits bekannt und beschrieben sind. Die *function junction* and *expression connection* Schnittstellen von SGD [Ce02] sind gute Beispiele für diese Art von Diensten. Nur wenige Dienste, wie EPCLUST/ EP:PPI [Ke02B][BV00], ermöglichen es dem Nutzer selbst, die Ergebnisse verschiedener experimentelle Studien miteinander zu kombinieren. Im Ergebnis erhält er eine Liste von Gennamen, welche die Anforderungen seiner Anfrage erfüllen. Die Weiterverarbeitung dieser Daten im Sinne einer Interpretationsunterstützung und Navigation wird nicht angeboten. Daher beklagen die Nutzer in den Lebenswissenschaften generell, dass sie zu wenig Unterstützung in Form integrierter Analyseumgebungen und Softwarewerkzeuge erhalten.

2. Vorgehensweise

Der in diesem Artikel beschriebene Ansatz, der sich während der letzten zwei Jahre herauskristallisiert hat, unterstützt den Wissensgewinn aus Biodaten. Auf der Grundlage von Experimentaldaten, die aus allen oben genannten Datenkategorien herangezogen werden, kann eine effiziente, interessengeleitete Datendurchsicht nach impliziten Zusammenhängen vorgenommen werden. Sowohl technische als auch theoretische Aspekte dieser Vorgehensweise werden in diesem Abschnitt beschrieben und diskutiert.

2.1 Datenmodellierung

Das Konzept der Datenverwaltung basiert auf einem relationalen Datenbanksystem. Dessen Entity-Relationship Modellierung orientiert sich sowohl an den vorhandenen Datensätzen sowie deren Zugehörigkeit zu einer der drei Datenkategorien (siehe Abb. 1 für einen schematischen Überblick). An zentraler Position des Datenmodells steht ein Basismodell, welches als Bindeglied zwischen den oben genannten Datenkategorien fungiert. Es orientiert sich am zentralen Dogma der Biologie und beschreibt u.a. Gene, Genvarianten, Proteine (*gene*, *gene_variant*, *protein*) und deren Beziehungen untereinander in ihrer zellulären Umgebung. Die gewählte Modellierung erlaubt es, jeden Datensatz eindeutig seinem biologischen Kontext zuzuordnen und gewährleistet damit eine sinnvolle Verknüpfung der einzelnen Submodelle (siehe Abb. 1, Submodelle gruppiert um ein zentrales Basismodell).

[1] D.h. sogenannte Hyperlinks werden gelegt, wenn *irgendein* Bezug zwischen zwei Dokumenten konstruiert werden kann.

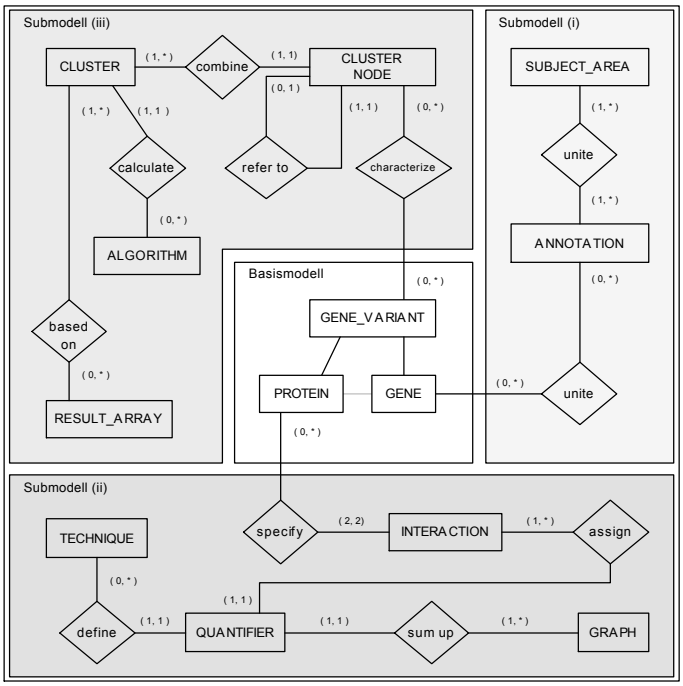


Abb. 1: Schematische Darstellung des gewählten Datenmodells

Zum besseren Verständnis wird in der folgenden Erläuterung eine konkrete Ausprägung (Genomforschung) des Datenbankmodells herangezogen. Ein oft wiederkehrendes und zentrales Problem der Datenintegration auf dem Gebiet der funktionellen Genomforschung besteht in den divergierenden Nomenklaturkonventionen[2]. Bei Vernachlässigung dieses Faktums bieten sich jedoch folgende Submodelle (orientiert an den erwähnten Datenkategorien) für die Datenintegration an. Der Bezug zum Datenmodell wird über die Angabe der Entitäten (kursiv) gewährleistet und wird für jede Datenkategorie einzeln dargestellt:

(i) Die Zuordnung von Eigenschaften zu Objekten (im Beispiel Gene/Genprodukte) erfolgt über ein festgelegtes Vokabular (*annotation*), welches in einzelne Themenkreise (*subject_areas*) gegliedert ist. Dieses Vorgehen sichert aus Sicht der Inhaltsverwaltung neben der redundanzfreien Verwaltung auch eine klare Unterscheidung zwischen den Objekten und Konzepten ab, denen die Eigenschaften zugeordnet sind (Abb. 1, Submodell (i)).

(ii) Eine Beziehung (*interaction*) zwischen zwei Objekten wird mit einem Quantitätsmaß (*quantifier*) und einem Methodenindikator (*technique*) spezifiziert. Der

[2] Die korrekte biologische Zuordnung für ein gegebenes Attribut (SWISSPROT z.B. unterscheidet in der Regel nicht nach Spleißformen bei der Angabe der Gewebe, in welchen das Protein als exprimiert gefunden wurde.) muß zumeist manuell vorgenommen werden, da bis zum heutigen Zeitpunkt keine generische, computergestützte Lösung abzusehen ist.

Methodenindikator bezeichnet dabei die Methode oder Technologie, mit der die Beziehung erkannt bzw. vorhergesagt wurde. Die Interpretation des Quantitätsmaßes, soweit angegeben, ist nur im Kontext der verwendeten Methode gültig. Dessenungeachtet stellt sie einen eindeutigen Bezugspunkt zu jeder der beteiligten ternären Beziehungsentitäten *interaction*, *technique*, und *graph* dar. Für die Verwaltung innerhalb des Datenmodells ist es jedoch unwesentlich, inwieweit der aus den paarweisen Beziehungen entstehende Graph (*graph*) gerichtet oder ungerichtet ist. Es bleibt den Algorithmen zum Abgriff und zur Weiterverwendung überlassen, die korrekte Interpretation sicherzustellen (Abb. 1, Submodell (ii)).

(iii) Eigenschaftszuordnung können sich gleichermaßen auf eine Menge von Objekten beziehen, die ihrerseits eine biologische Konzeptebene bzw. ein Partitionierungsprinzip in Form von Clustern[3] begründen können. Die Modellierung der Clusterbäume (*cluster*) erfolgt auf der Grundlage von Experimentaldaten (*result_array*). Parallel wird die hierarchische Struktur der einzelnen Cluster (*cluster_node*) in Verbindung zu den beteiligten Objekten (*gene_variant*) verwaltet. Mit dem Einsatz von Algorithmen oder Beschreibungsprozeduren (*algorithm*) kann zudem das rechnerische Vorgehen belegt werden. Die gewählte Informationsstruktur des Datenmodells lässt jedoch keine Aussagen über die Auswahl oder Abfolge von Objekten innerhalb eines Clusters zu. Mathematisch gesehen sind Cluster demnach schlicht Mengen von Objekten (Abb. 1, Submodell (iii)).

2.2 Datenaufbereitung

Nachdem beschrieben wurde, wie die Datensammlungen in der Datenbank verwaltet werden, wird sich im nächsten Schritt der Fragestellung zugewandt, wie ein Nutzer Datensätze für seine spezifischen Anfragen kombinieren kann. Die Datenaufbereitung orientiert sich einerseits an vordefinierten Formatvorlagen, die eine manuelle Zusammenstellung der Datensätze gestatten und wird anderseits von der Anwendung *DataReader* unterstützt. Letzterer verbindet sich via ODBC mit der Datenbank und ermöglicht es so, eine interaktive Auswahl aus allen vorhandenen Datensätze zu treffen.

Zur Datenvisualisierung und -analyse wird ein induzierter Graph verwendet, welcher auf der Grundlage der ausgewählten Datensätze gebildet wird. Mit der Zusammenstellung des Graphen wird demnach eine erste Einschränkung der Datensätze auf die zu betrachtenden Fragestellung vorgenommen.

[3] In dieser Arbeitbesiehen sich alle Textpassagen auf hierarchische Clusterbäume, ungeachtet dessen, inwieweit es sich dabei um gewichtete Cluster handelt oder nicht.

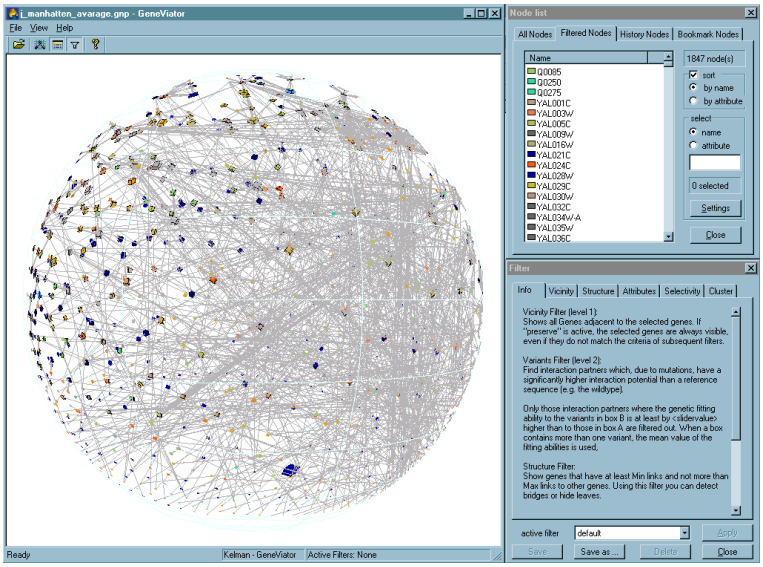


Abb. 2: Screenshot der GeneViator Anwendung

Um alle zum heutigen Zeitpunkt existenten und in Zukunft als realistisch anzusehenden Größenordnungen von Datensätzen zu unterstützen, bedient sich das Visualisierungsprinzip des GeneViator (siehe Abb. 2) der hyperbolisch verzerrten Projektion von Graphenstrukturen in einer Kugel, wie sie von Tamara Munzner als H3-API [Mu00][Ke02A][Hy02] entwickelt bzw. erweitert wurde. Dieser Ansatz erlaubt es, einen theoretisch unendlichen Satz von Knoten auf einem typischen Computerbildschirm darzustellen. Jeder Knoten des Graphen spiegelt ein Objekt des Gegenstandsbereiches wieder, welche über eine/mehrere Beziehung(en) miteinander verbunden sein können. Nachfolgend wird erläutert, wie die einzelnen Datenkategorien zur Erzeugung des Graphen genutzt werden[4].

(i) Die Datensätze der Kategorie (i) werden den Knoten als Attributliste zugeordnet. Jedem Attribut steht bei der späteren Visualisierung genau ein Farbwert gegenüber. Die Kombination der Attribute eines Knoten spiegelt sich demnach in der entsprechenden Färbung jedes einzelnen Knotens wieder. Vorläufig geschieht die Farbzuweisung automatisch und kann vom Nutzer nicht beeinflusst werden.

(ii) Die Kanten des Graphen werden aus den gewählten Datensätzen der Kategorie (ii) erzeugt, d.h. jede vorhandene paarweise Beziehung wird als eine Kante zwischen den beteiligten Knoten wiedergegeben. Sollten mehrere Kanten zwischen zwei Knoten auftreten, wird aus Gründen der Übersichtlichkeit nur eine Kante dargestellt. Sind im Anschluss daran nicht alle Knoten in einem zusammenhängenden Graphen verbunden, wird ein virtueller Wurzelknoten eingefügt, der alle entstandenen Subgraphen über Kanten verbindet. Das ist dem Fakt geschul-

[4] Ein illustrierendes Beispiel wird im Abschnitt Anwendungen dargestellt.

det, dass die H3-API wenigstens einen Wurzelknoten für die Verarbeitung von Wäldern[5] benötigt.

(iii) Die Datensätze der Kategorie (iii) besitzen keinen Einfluss auf die Struktur des Graphen, können jedoch genutzt werden, um mittels Filterfunktionen Teile des Graphen auszublenden. Zusammenfassend spiegeln die gewählten Datensätze in ihrer Kombination eine, von der Datenbank losgelöste statische Datensammlung wieder, die zur Visualisierung des Graphen im GeneViator [He02] verwendet wird.

2.3 Datenverarbeitung

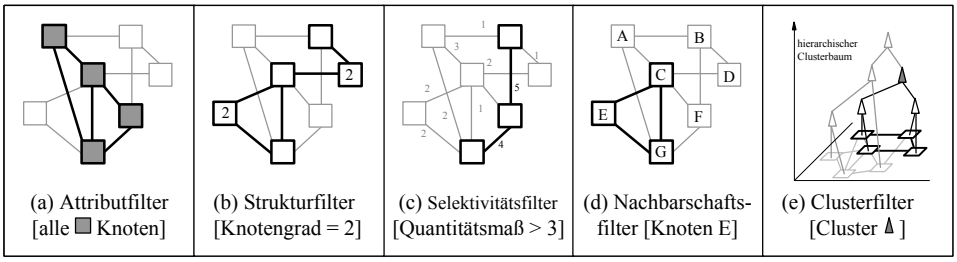
Auf Grundlage der in Abschnitt 2.2 erläuterten Datenzusammenstellung können Experimentaldaten aller drei Datenkategorien im GeneViator untersucht werden. Diese ermöglichen nicht nur eine willkürliche Auswahl der Datensätze unabhängig ihrer Kategoriezuordnung, sondern erlauben zudem die interaktive Navigation innerhalb der Datenrepräsentation und bieten Strategien zur Datenanalyse an. Die implementierten Methoden der Datenanalyse zielen darauf ab, eine breite Spanne von Nutzerstrategien zu unterstützen, die für das Aufdecken von wissenschaftlich relevanten Korrelationen innerhalb des untersuchten Systems adäquat erscheinen.

Für alle drei Datenkategorien wird die Filterung in zwei Richtungen unterstützt (siehe Abb. 3), zum einen *top-down* (Ausgehend von der Eigenschaften) zum anderen *bottom-up* (Ausgehend von den Objekten). Dieses Vorgehen verfolgt die Zielstellung den aufgabenspezifischen Anforderungen bestmöglich zu genügen. Einschränkungen bezüglich der Analysekomplexität müssen lediglich im Bereich der filterübergreifenden 'oder' bzw. 'und/oder nicht ' Verknüpfungen der gewählten Kriterien hingenommen werden. Für die nachstehende Erläuterungen der Analysesichtweise wird das beschriebene graphentheoretische Vokabular beibehalten.

	Datenkategorie (i)	Datenkategorie (ii)	Datenkategorie (iii)
top-down Vorgehen	Attributfilter	Strukturfilter Selektivitätsfilter	Clusterfilter
bottom-up Vorgehen	-	Nachbarschaftsfilter	Clusterfilter

Abb. 3: Tabelle der Filterstrategien des GeneViator

[5] Die H3-API basiert auf Bäumen, d.h. azyklischen Graphen. Graphen, die aus nicht verbundenen Bäumen bestehen, heißen Wälder. Kanten, die Zyklen induzieren, werden intern anders gehandhabt, das ist jedoch nicht Thema der Darlegungen in dieser Arbeit.



(a) Attributfilter [alle ■ Knoten] | (b) Strukturfilter [Knotengrad = 2] | (c) Selektivitätsfilter [Quantitätsmaß > 3] | (d) Nachbarschaftsfilter [Knoten E] | (e) Clusterfilter [Cluster ▲]

Abb. 4: Skizzierte Darstellung der Filterstrategien des GeneViator am einfachen Beispiel

Das *bottom-up* Vorgehen unterstützt den knotenorientierten (im Beispiel Gen getriebenen) Prozess der Suche nach Korrelationen. Durch Festlegung einer Knotenauswahl als Anfrage kann der Nutzer untersuchen, welche biologischen Konzepte (auch Kombinationen solcher) seine spezifische Auswahl abdecken. Die biologisch-konzeptuell getriebene Suche nach Korrelationen hingegen wird durch das *top-down* Vorgehen unterstützt. Der Nutzer wählt biologische Konzepte, die für ihn von Interesse sind, und bestimmt so Teilmengen von Knoten (im Beispiel Gene), die seinen vorgegebenen Kriterien entsprechen.
Im folgenden wird beschrieben, wie diese Vorgehensweisen auf die beschriebenen Datenkategorien angewandt und kombiniert werden können. Außerdem wird aufgezeigt, wie die entstehende Komplexität mit dem Visualisierungs- und Navigationswerkzeug GeneViator beherrscht werden kann:

Datenkategorie (i)
Für die Auswahl von Knoten anhand ihren Eigenschaften ist die *top-down* Vorgehensweise als der spezifische Attributfilter (siehe Abb. 4a) implementiert. Es kann eine frei zu wählende Kombination von Attributen festlegen werden. Die graphische Repräsentation des Gesamtgraphen reduziert sich daraufhin auf diejenigen Knoten (im Beispiel Gene/Genprodukte), die wahlweise mindestens einem der Attribute oder allen ausgewählten Attributen genügen.

Datenkategorie (ii)
Wie eingangs erwähnt werden paarweise Beziehungen als Kanten zwischen den betreffenden Knoten dargestellt. Diese graphenorientierte Sichtweise lässt beide oben genannten Vorgehensweisen zur Datenanalyse sinnvoll erscheinen. Das *top-down* Vorgehen wird durch zwei Filtertypen unterstützt. Der Strukturfilter (siehe Abb. 4b) ermöglicht es, den induzierten Graphen auf Subgraphen mit einem definierten minimalem und/oder maximalem Knotengrad[6] zu reduzieren. Wenn die untersuchten Kanten zudem mit einem quantitativen Maß (siehe Abschnitt 2.1) versehen sind, kann der Graph auch auf Kanten oberhalb einer einstellbaren minimalen Stärke weiter reduziert werden. Diese Stellgröße wird über den implementierten Selektivitätsfilter (siehe Abb. 4c) geregelt. Unterstützung für das *bottom-up* Vorgehen bietet der Nachbarschaftsfilter (siehe Abb. 4d). Der Nutzer kann eine beliebige Gruppe von Anfrageknoten (im Beispiel Gene) auswählen und sich dadurch selektiv diejenigen Knoten einblenden, die zu einem belie-

[6] Anzahl der Nachbarn eines Knotens

bigen der Anfrageknoten benachbart sind. Wiederholte Anwendung dieses Filters führt zur Aufdeckung weiterer Nachbarschaftskreise, die entsprechend über 1 bis n-1 Kanten (wobei n für die maximale Anzahl von Knoten in einer Abfolge steht) erreichbar sind.

Datenkategorie (iii)
Unterstützung beim Navigieren in und Filtern von mengenwertigen Experimentaldaten (z.B. hierarchisch strukturierte Clusterbäume) bietet die jüngste Funktionalitätserweiterung der Clusterfilter. Dieser ermöglicht es einen/mehrere Clusterbäum(e) dem Graphen orthogonal gegenüberzustellen (siehe Abb. 4e). Jeder Knoten des Graphen kann somit gleichzeitig einen Blattknoten innerhalb eines oder mehrerer Clusterbäume darstellen. Der *top-down* orientierte Zugriff verfolgt eine geradlinige Strategie der Clusterauswahl, d.h. der Nutzer wählt zunächst den/die für ihn relevanten Cluster auf der entsprechenden Hierarchieebene des Clusterbaums aus. Der GeneViator reduziert den Graphen anschließend auf diejenigen Knoten, die als Blattknoten unterhalb dem/den ausgewählten Cluster(n) liegen. Die *bottom-up* Vorgehensweise stellt sich dagegen komplexer dar[7], hierbei trifft der Nutzer im ersten Schritt eine Auswahl der Knoten des Graphen, die im weiteren als Anfragemenge (Abb. 5, "A, B, G, H") über einem oder mehreren Clusterbäumen genutzt werden können. Um die folgende Identifikation von (hoffentlich) bedeutsamen Clustern auf möglichst tiefer Hierarchieebene des Clusterbaums zu unterstützen, wurden zwei Suchparameter eingeführt: *Trennschärfe* und *Unterstützung*. Mit Hilfe der *Trennschärfe* kann die Suche auf eine gegebene minimale und maximale Mächtigkeit, d.h. die zulässige Anzahl der Blattknoten unterhalb eines Clusters einschränkt werden. Für das in Abb. 5 gewählte Beispiel (Trennschärfe = [3, 5]) bedeutet dies, dass mindestens 3 und maximal 5 Blattknoten in einem Cluster enthalten sein müssen, damit dieser ausgewählt wird. Die Wahl der *Trennschärfe* ermöglicht es damit, indirekt die Hierachieebenen von Interesse[8] einzugrenzen. Der Parameter *Unterstützung* spezifiziert, wieviele der ausgewählten Anfrageknoten mindestens in einem Cluster enthalten sein müssen, damit dieser als "Treffer" angezeigt wird. Das in Abb. 5 gewählte Beispiel (Unterstützung = 2) impliziert, dass mindestens zwei Knoten der Anfrageliste im Cluster enthalten sein müssen, damit dieser ausgewählt wird. Der verwendete Algorithmus wurde dahingehend optimiert, dass ein nachträgliches Erweitern bzw. Ausdünnen der Anfrageknoten lediglich eine Berechnung der veränderten Komponenten nach sich zieht. Diese Strategie macht Sinn, wenn bedacht wird, dass speziell im Forschungsbereich der Autoren (Genom-, Transkriptom- und Proteomforschung) eine iterative Annäherung an die endgültige Anfrageformulierung realistisch ist. Es ist demnach ausdrücklich erwünscht, dass ein Nutzer interaktiv die Einstellungen der getroffenen Knotenanfrage variiert, bis er auf Cluster stößt, die sein Interesse auf sich ziehen. Das wird in den Fällen gelingen, wenn eine intuitiv postulierte (vermutete) Korrelation auch tatsächlich hinter den Daten versteckt ist.

[7] Eine naive Implementation für Hierarchien hätte immer ein triviales Ergebnis, nämlich, daß die oberste Hierachieebene (Wurzelknoten) alle in die Untersuchung einbezogenen Blattknoten umfaßt.
[8] Die Mächtigkeit wurde gegenüber der Ebene bevorzugt, weil die Auswertung nicht auf balancierte Hierarchien einschränkt werden sollen. Besonders für tief gestaffelte binäre Hierarchien erlaubt dies eine intuitivere Kennung.

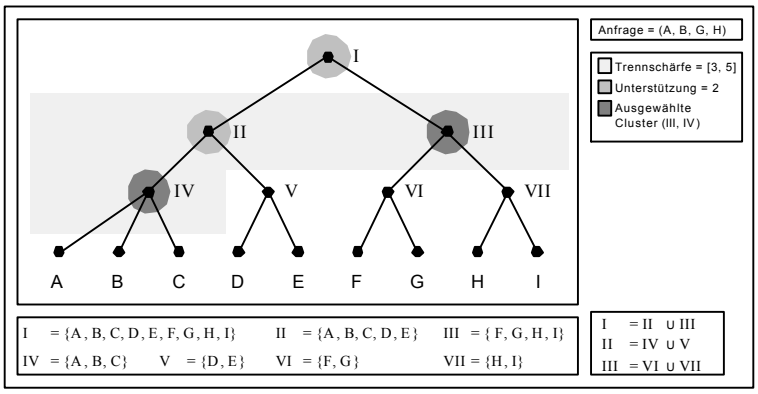


Abb. 5: Einschränkungen einer hierarchischen Suchabfrage
mit den Parametern *Trennschärfe* und *Unterstützung*

Wie oben ausgeführt, können alle beschriebenen Filtervorgänge in beliebiger Reihenfolge und Kombination ausgeführt werden. Dadurch ist eine variable Untersuchung eines interessierenden Systems, gerade in Bezug auf die Überlagerung unterschiedlicher Informationsdimensionen gewährleistet. Angesichts der Komplexität des Forschungsprozesses und bei gegebener Datenlage kann jedoch nicht von vornherein garantiert werden, ob die Daten überhaupt Korrelationen beinhalten und erkennen lassen. Um dem Nutzer eine verlässliche Führungshilfe durch multidimensionale Datensammlungen zu geben, haben die Autoren eine Art "Lackmustest" für eine gegebene Datenkombination entwickelt, der im Abschnitt Anwendung dargelegt wird.

3 Anwendung

Nachdem die vollständige Bestimmung aller Gensequenzen einer Spezies zur wissenschaftlichen Alltäglichkeit geworden ist, besteht nunmehr die Herausforderung, diesen Genen ihre Funktionen in den Lebensprozessen zuzuordnen. Hochdurchsatzexperimenten, in denen jeweils ein molekularbiologischer Aspekt für das gesamte Genom erhoben wird, sollen die dafür notwendigen Daten liefern. Als besonders vielversprechend für die Funktionsaufklärung gelten derzeit Untersuchungen zur sogenannten Genexpression und zur Proteinwechselwirkung. Die Genexpression beschreibt, welche Gene in einem bestimmten Zusammenhang gemeinsam aktiviert sind. Proteinwechselwirkungsstudien versuchen, systematisch zu erheben, welche Genprodukte in ihrem unmittelbaren Zusammenspiel Lebensfunktionen übernehmen können. Die Daten, die im Rahmen dieser Verfahren bestimmt werden, sind besonders geeignet, den Nutzen des hier beschriebenen Ansatzes zu demonstrieren. Zum einen ist es erforderlich, dass zwei Gene, deren Produkte gemeinsam eine Funktion erfüllen, auch zur gleichen Zeit aktiv sind. Zum anderen hat sich gezeigt, dass klassische, statistische Verfahren nicht vermögen, die Korrelationen innerhalb einer einzigen Verfahrensweise aufzudecken [KS02], [Ku02].

Die Bäckerhefe ist die Spezies, für die derzeit die umfangreichsten und vollständigsten Datensammlungen zu den beschriebenen Fragestellungen verfügbar sind. In der hier beschriebenen Anwendung wird gezeigt, wie ausgehend von den Genomdaten der Hefe, Korrelationen in systematisch zusammengetragenen Experimentaldaten aufgedeckt werden können.

3.1 Datenintegration

Entsprechend der in Abschnitt 2 dargelegten Vorgehensweise sind die notwendigen Daten modelliert, zusammengestellt und in eine Datenbankinstanz[9] integriert worden:

Das Bezugsystem der Analyse bildet das Hefegenom, so wie es in der *Saccharomyces Genome Database* (SGD) [Ce02] bereitgestellt wird. Dort sind alle bekannten Hefegene mit ihren jeweiligen Sequenzen, Namen und eindeutigen Bezeichnungen zusammengestellt. In der Hefeforschung haben sich die sogenannten ORF-Namen als eindeutige Genbezeichner durchgesetzt. Ein ORF (*open reading frame*) bezeichnet einen Abschnitt des Hefegenoms, der die Bildungsvorschrift für das dem Gen zugeordnete Genprodukt enthält. Für die einzelnen Datenkategorien wurden folgende Datenquellen verwendet:

(i) Zur Charakterisierung von Genen wird die *Gene Ontology* (GO) verwendet [As00]. GO stellt Begriffshierarchien für die Beschreibung von *biologischen Prozessen, Zellkomponenten* und *molekularen Funktionen* bereit. Die Charakterisierung eines Gens erfolgt, indem einem ORF-Namen der jeweils spezifischste Begriff aus den drei Hierarchien zugeordnet wird. Damit ergibt sich im Idealfall ein Begriffstripel, das einem ORF-Namen als Attribut zugeordnet wird. Auch wenn diese Zuordnungen noch im Wesentlichen als *work in progress* bewertet werden müssen[10], kann für die Hefe in Anspruch genommen werden, dass aufgrund der intensiven internationalen Forschungsbemühungen die Fehlerquote im Vergleich zu anderen Spezies als gering einzuschätzen ist.

(ii) Binäre Beziehungen zwischen Genen werden aus der 'Physical Interaction Table' abgeleitet, die von der *Comprehensive Yeast Genome Database* (CYGD) [Me02] verwaltet wird. In dieser Tabelle werden weltweit publizierte Daten zu experimentell nachgewiesenen Proteinwechselwirkungen zusammengestellt[11]. Ein Eintrag in dieser Tabelle besteht aus zwei ORF-Namen und der Methode, mit der eine Wechselwirkung festgestellt wurde.

(iii) Mengenmäßige Attributzuschreibungen wurden aus einer Studie mit dem Namen *Yeast Cell Cycle Analysis Project* aufbereitet, die von Spellman et al. 1998 publiziert worden ist [Sp98][Ce02]. In dieser Studie wurde die Genexpression der Hefe im Zusammenhang mit dem Zellzyklus bestimmt, den eine Zelle zwischen zwei Teilungen durchläuft.

---

[9] DB2 UDB V7.2 auf einem Linux Server

[10] Häufig ist einem ORF-Namen auch zugeordnet, dass über Prozess, Funktion und Lokalisierung keine Aussage gemacht werden kann.

[11] Die Tabelle enthält zudem Wechselwirkungen zwischen Proteinen und funktionalen RNA Molekülen. Diese wurden in unserer Anwendung nicht berücksichtigt.

Abbildung 1 zeigte das Datenmodell, in das die Daten integriert wurden. Dabei konnten 1.892 der ca. 6.300[12] Hefegene, Daten aller beschriebenen Kategorien zugeordnet werden. Diese Gene bilden die Basis für die hier dargestellten Analysen. Die hierarchische Clusteranalyse der Expressionsdaten wurde mit dem Werkzeug J-Express [Mo02] durchgeführt.

3.2 Suche nach Korrelationen

Das Explizieren von Korrelationen, die in den experimentellen Daten vorhanden sind, ist eine wichtige methodische Voraussetzung für das Verständnis des Zusammenspiels der Gene in lebenden Organismen. Wie bereits erwähnt, sind die bekannten statistischen Verfahren zur Korrelationsanalyse jedoch häufig nicht geeignet, die wesentlichen Zusammenhänge aufzudecken. Dies ist insbesondere in dem Umstand begründet, dass kein linearer Zusammenhang zwischen den wahren Werten und den Beobachtungswerten einer Größe, die experimentell erfasst werden soll, unterstellt werden kann. Insbesondere das Protokoll zur Erfassung von Expressionsdaten umfasst mehrere Schritte, in denen jeweils nicht lineare Transformationen der Beobachtungsgröße erfolgen[13]. Bei der Erfassung von Proteinwechselwirkungsdaten muss erheblich in das zu beobachtende System eingegriffen werden. Daher muss regelmäßig angenommen werden, dass sich nicht die „natürlichen Verhältnisse“ einstellen können. Dennoch wird allgemein angenommen, dass die verfügbaren Beobachtungsdaten zumindest qualitativ ein zutreffendes Bild der untersuchten Lebensprozesse liefern. Derzeit gibt es weltweit Anstrengungen, eine Vielzahl von Data-Mining-Methoden für die Analyse von Genomdaten nutzbar zu machen. Stellvertretend seien hier das *frequent subgraph mining* [KK01] und der Aufbau von *Bayesian networks* [Pe01] genannt, die erste, vielversprechende Ergebnisse geliefert haben. Das *frequent subgraph mining* ist aus Effizienzgründen auf Probleme moderater Größe beschränkt. Das Ableiten von *Bayesian networks* aus Expressionsdaten erfordert für jeden Anwendungsfall das Aufstellen komplexer wahrscheinlichkeitstheoretischer Modelle, die einem standardmäßigen Einsatz der Methode im Wege stehen. Wir schlagen daher im Folgenden eine Vorgehensweise vor, die zumindest qualitativ eine Aussage über die Korrelation von Genen ermöglicht.

Gesucht werden also Korrelationen zwischen Objekten, in diesem Falle Gene, für die Beobachtungsdaten unterschiedlicher Art vorliegen. Wir werden diese Fragestellung im Folgenden dahingehend interpretieren, dass für eine Teilmenge der Gene entschieden werden soll, ob sie hinsichtlich mehrerer Kriterien in ein gemeinsames Klassifikationsraster fallen. Die Vorgehensweise lehnt sich an die *bottom-up*-Navigation in Mengenattributen an, so wie sie in Abschnitt 2 beschrieben wurde. Dabei werden folgende Beobachtungen für die Entscheidung herangezogen:

---

[12] Die exakte Anzahl von Hefegenen ist immer noch nicht bekannt. Jedoch werden ca. 6.300 ORF-Namen von mehreren Datenbanken seit Jahren als gesicherte Gene verwaltet.

[13] Würden diese Transformationen für alle gleichzeitig beobachteten Gene gleichförmig sein, wäre das kein prinzipielles Problem. Jedoch wirken sich die Prozesse in Abhängigkeit von der Länge und Nukleotidsequenz einzelner Genen (nicht vorhersagbar) unterschiedlich auf die Messwerte aus.

1. Beobachtung:
Werden Gene ausgewählt, die hinsichtlich der Genexpression ähnliche Merkmalsausprägungen aufweisen, finden sich viele Cluster, die auf niedriger Stufe in der Hierarchie bereits mehrere Gene aus der Auswahl enthalten. Diese Cluster, die bei einem niedrigen Wert des Filterparameters *Unterstützung* gefunden werden, enthalten darüber hinaus nur wenige zusätzliche Gene, die nicht in der Auswahlmenge vorhanden sind. Das führt insgesamt dazu, dass bei niedrigem Wert für *Unterstützung* der Anteil der Gene aus der Auswahl größer ist als der Anteil von Genen die insgesamt angezeigt werden. Erhöht man ausgehend von dieser Situation die geforderte *Unterstützung* stufenweise, so gelangt man zu einem Punkt, an dem nur noch in einer vergleichsweise hohen Hierarchiestufe die Auswahlanforderung erfüllt wird und zwar nur noch für ein einzelnes Cluster. Daher gibt es an dieser Stelle einen Knick im Kurvenverlauf. Ab dieser Stelle finden sich dann alle gewählten Gene im gleichen Teilbaum der Hierarchie. Der Anteil der Gene aus der Auswahl, die sich in dem jeweiligen Cluster finden, ist stets größer als der Anteil der Gene, die insgesamt angezeigt werden.

2. Beobachtung:
Wählt man Gene aus, deren Expressionsverhalten keine Ähnlichkeit aufweist, so findet man zwar für niedrige *Unterstützung* eine Reihe von Clustern, die diese Gene enthalten, aber bereits hier werden viele Gene mit angezeigt, die nicht in der Auswahl vorhanden sind. Erhöht man die geforderte Unterstützung, dann „springen“ die gefundenen Cluster in der Hierarchie und die Anzahl der gefundenen Cluster kann sich bei jeder Erhöhung der Anforderung ändern. Daher kommt es zu einem oszillierenden Kurvenverlauf. Ab einem gewissen Punkt findet sich auch hier nur noch ein Teilbaum in der Hierarchie, in dem durch stetiges „Hochklettern“ die Anforderungen erfüllt werden. Jedoch ist in diesem Fall der Anteil der insgesamt angezeigten Gene stets größer als der Anteil der Gene aus der Auswahl, die im jeweiligen Cluster vorhanden sind.

In Abbildung 6 werden diese Beobachtungen anhand schematischer Kurvenverläufe dargestellt. Die gestrichelte Linie (aufgetragen gegen die rechte Y-Koordinate) gibt an, wie viele Gene aus der Auswahl in das Klassifizierungsraster fallen. Die durchgezogene Linie (aufgetragen gegen die linke Y-Koordinate) gibt an, wie viele Gene der Grundgesamtheit bei der jeweiligen Filtersetzung insgesamt zur Anzeige gelangen. Da beide Y-Koordinaten prozentual skaliert sind, treffen sich also beide Kurven in dem Punkt der vollständigen Abdeckung der jeweiligen Gesamtheit.

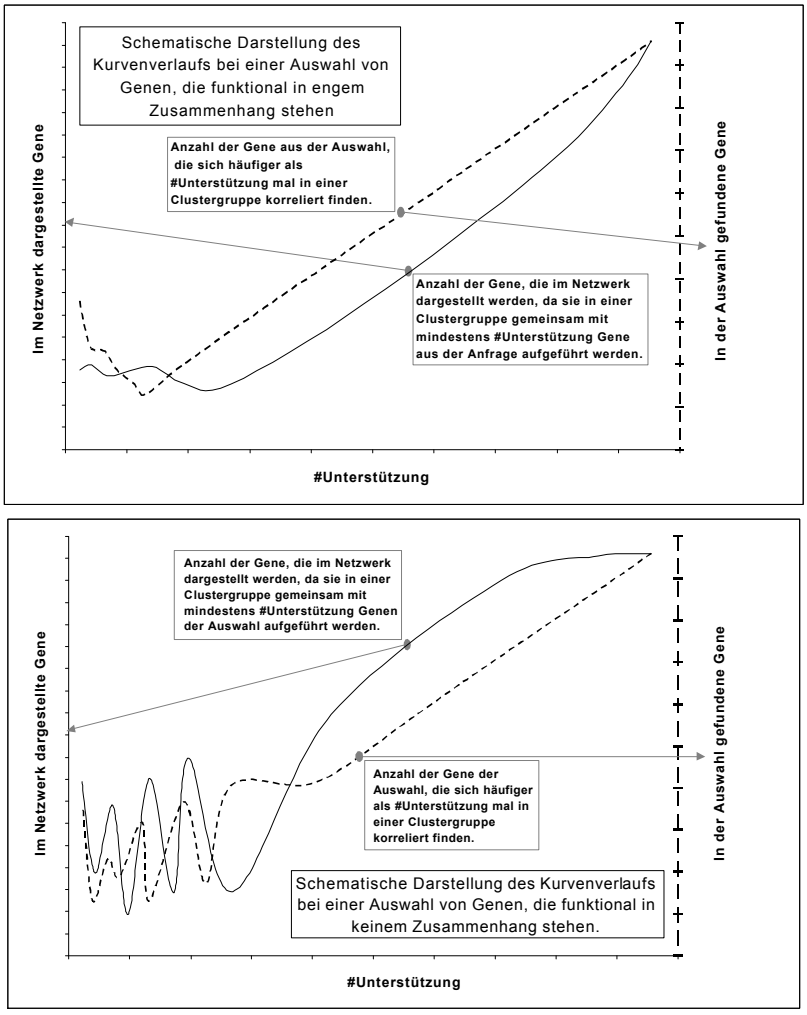


Abb. 6: Schematische Wechselbeziehung bei
a) korrelierter und b) nicht korrelierter Genauswahl

Die beschriebenen Beobachtungen lassen sich nun auf die allgemeine Auswahl von Genen übertragen: wählt man Gene basierend auf Kriterien anderer Kategorien, so kann man aus den analog abgeleiteten Kurvenverläufen schließen, ob diese Gene auch im Hinblick auf ihre Expression eine Beziehung aufweisen. Um zu prüfen, ob diese Schlussfolgerung einer Überprüfung anhand der integrierten Beobachtungswerte standhält, wurden Szenarios festgelegt, in denen bereits durch die Fragestellung die Entscheidung für oder wider Korrelation induziert wird.

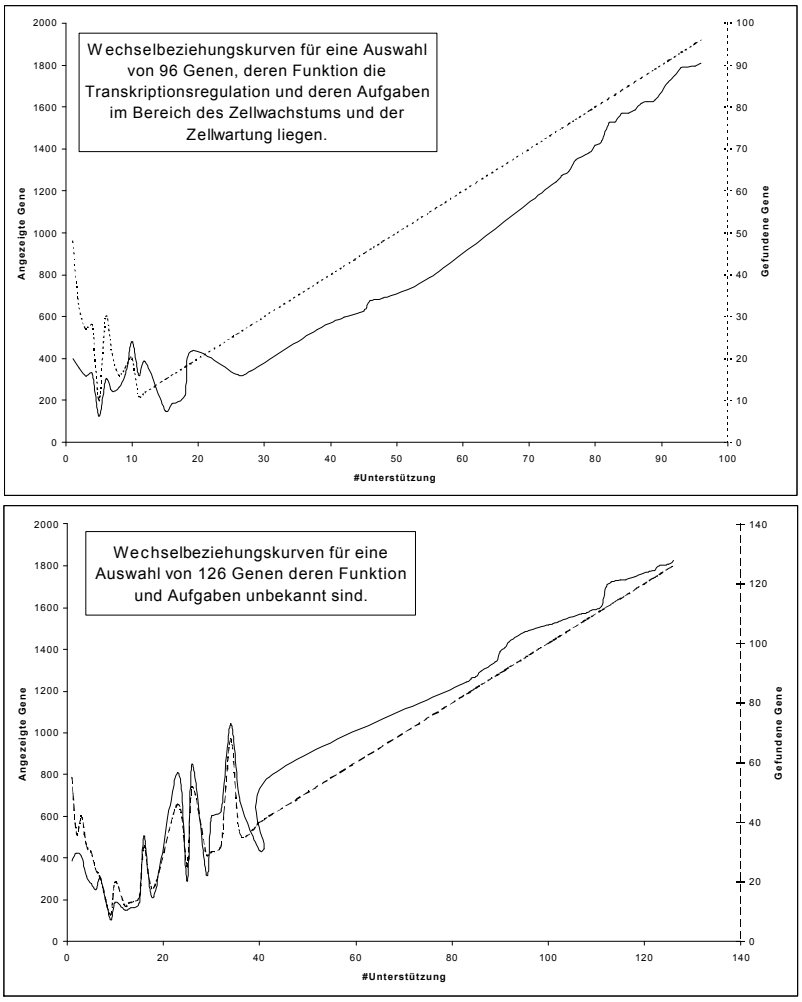


Abb. 7: Wechselbeziehung für
a) funktionell verwandte Gene und b) Gene unbekannter Funktion

Im **ersten Szenario** wurden alle Gene ausgewählt, die eine bekannte Rolle im *Zellwachstum und/oder bei der Zellwartung* spielen und außerdem die Funktion eines *Transkriptionsregulators*[14] ausüben. Diese Auswahl (auf Basis der Datenkategorie i) umfasst 96 Gene. Von diesen kann mit hoher Sicherheit angenommen werden, dass sie über ähnliche Expressionsmuster verfügen, da in den zu Grunde liegenden Messungen insbesondere die Genaktivität in Abhängigkeit vom Zellzyklus erfasst wurde.

Im **zweiten Szenario** wurden anhand der GO-Charakterisierung Gene ausgewählt, für die sowohl die *molekulare Funktion* als auch der *biologische Prozess unbekannt* sind.

[14] D.h. die Genprodukte dieser Gene regulieren das Aktivitätsniveau anderer Gene

Die Auswahl enthält 126 Gene, von denen mit hoher Sicherheit anzunehmen ist, das sie kein übereinstimmendes Expressionsmuster aufweisen.

Die für Szenario eins und zwei erstellten Kurvenverläufe sind in Abbildung 7 dargestellt und weisen im wesentlichen die Charakteristika auf, die in Beobachtung eins bzw. zwei dargelegt wurden. Insbesondere der Umstand, dass der Anteil der insgesamt angezeigten Gene unterhalb bzw. oberhalb des Anteils der *unterstützten* ausgewählten Gene liegt, zeichnet sich als geeignetes Entscheidungskriterium für oder wider Korrelation ab.

Diese Beobachtung konnte auch in weiteren Szenarios untermauert werden, in denen z.B. Gruppen von Genen ausgewählt wurden, die über Daten der Kategorie ii vernetzt waren (Proteinwechselwirkung) bzw. für die keinerlei Hinweis auf eine physische Interaktion vorlagen.

Abbildung 8 zeigt den Ausschnitt des komplexen Netzwerkes, der sich bei Verwendung der in Abschnitt 2 beschriebenen Filter auf eine konkrete molekularbiologische Fragestellung ergibt.

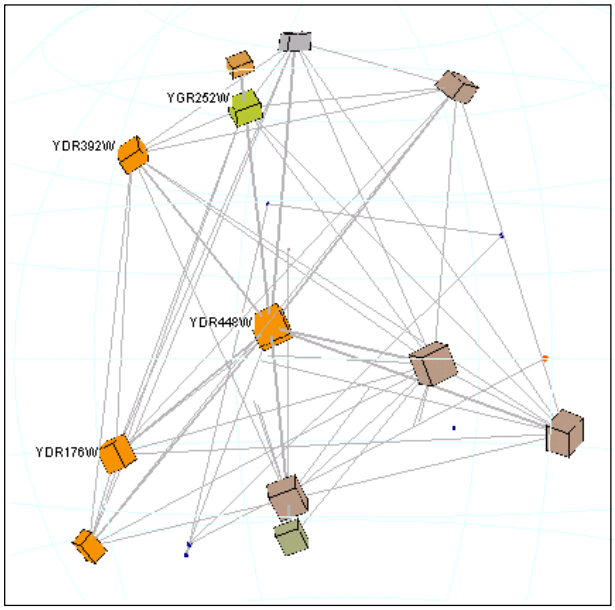


Abb. 8: GeneViator Screenshot einer bottom-up-Auswahl. Die Anfrage bildeten die physischen Wechselwirkungspartner der primären Stufe (19 Gene) und sekundären Stufe (51 Gene) des Hefegens YDR448W. Ausgehend von diesen wurden 27 Gene als korreguliert mit YDR448W identifiziert, übereinstimmend zu Spellman's Untersuchung [Sp98].

4 Zusammenfassung und Ausblick

Mit der vorliegenden Arbeit wird ein weitgehend werkzeuggestütztes Verfahren zur Suche nach Korrelationen in heterogenen Datensammlungen unterschiedlicher konzeptueller Kategorien vorgestellt. Das Vorgehen verfolgt den Zweck, versteckte und potentiell bedeutsame Korrelationen qualitativ festzustellen. Bei der Anwendung auf eine typische Zusammenstellung verschiedener Datenquellen, konnten die Machbarkeit und der Nutzen des Ansatzes gezeigt werden.

Derzeit erlaubt der Ansatz jedoch nur eine Überprüfung auf Korrelation in einer vom Anwender selbst erstellten Auswahlmenge. Es wäre wünschenswert, dass das System selbständig Gengruppen identifiziert, die vor dem Hintergrund der vorliegenden Beobachtungsdaten korreliert sind. Da jedoch prinzipiell eine in der Anzahl der Gene exponentielle Anzahl von Teilmengen existiert, müssen hier erst leistungsfähige Heuristiken für den Aufbau und die Auswahl potentiell interessanter Gengruppen entwickelt werden.

Weiterhin sind bereits neue Hochdurchsatzverfahren im Einsatz, die Ergebnisse liefern, die sich nicht ohne weiteres in eine der drei beschriebenen Datenkategorien einordnen lassen[15]. Wir streben an, auch diese Datensätze in unsere integrierte Verwaltungs- und Analyseumgebung einzubeziehen.

Glossar

Aminosäure	Grundbaustein der Proteine. Aminosäuren haben den gleichen chemischen Grundaufbau und unterscheiden sich voneinander durch die Beschaffenheit ihrer Seitenketten.
DNA	Desoxyribonukleinsäure, der materielle Träger der biologischen Erbinformation. DNA ist ein lineares Polymer aus vier *Nukleotiden* mit den Symbolen A, T, G, C in Doppelhelixform
Expressionshöhe	hier: Anzahl der Transkripte von einem Gen.
Gen	ein Abschnitt auf der *DNA* mit einer bestimmten funktionellen Bedeutung
Genetischer Code	Zuordnung von je drei aufeinanderfolgenden Nukleotiden zu einer *Aminosäure*. Die Gesamtheit der 4^3=64 möglichen Dreierkombinationen (Nukleotid-Triplets) und der zugeordneten Aminosäuren wird auch als Translationstabelle bezeichnet. In der Tabelle sind auch die Tripletts enthalten, die das Signal für Beginn bzw. Abbruch der Proteinsynthese vermitteln.
Gensequenz	die Abfolge der Nukleotide entlang eines Gens, in der Informatik als Zeichenkette aus den Nukleotid-Symbolen dargestellt

[15] Die experimentelle Feststellung von Proteinkomplexen mit Hilfe der Massenspektrometrie ist eines dieser Verfahren.

Genexpression	der Prozess der Realisierung des Informationsgehalts eines Gens. Wird ein Gen aktiviert, kommt es zunächst zur Anfertigung von n Kopien seiner Sequenz. Dieser Teilprozess heißt Transkription. Die Transkripte werden zurechtgeschnitten und als *mRNA* zum Ort der Proteinsynthese transportiert. Dort wird im Teilprozess der Translation die von der mRNA überbrachte Synthesevorschrift für ein *Protein* realisiert, indem je drei Bausteine nach dem *genetischen Code* festlegen, welche *Aminosäure* in das Protein eingebaut wird
Genom	die Vereinigungsmenge der Erbanlagen eines Organismus oder einer Spezies
mRNA	Messenger RNA, ein lineares Boten-Molekül, das die kodierende Information zum Ort der Proteinsynthese überträgt
Nukleotid	hier: Bausteine der linearen Polymere, die Erbinformation speichern bzw. transportieren
Offenes Leseraster	Abschnitt einer kodierenden Gensequenz zwischen Start- und Stoppkodon (s.a. Genetischer Code). Da die Einbaureihenfolge der Aminosäuren in ein Protein durch Tripletts festgelegt ist, ist die Länge Offener Leseraster ein Vielfaches von 3.
Protein	Lineares Polymer aus 20 *Aminosäuren*. Proteine bilden den Hauptteil der eigentlich operativen Bestandteile der Zelle. Sie bewirken Strukturbildung, Stoffwechsel, Energiehaushalt, Reproduktion und alle anderen wichtigen Lebensvorgänge in einer Zelle oder sind zumindest daran beteiligt.
Zentrales Dogma	Die *Genexpression* ist ein Prozess des horizontalen Informationsflusses (durchgezogene Pfeile) zwischen DNA, RNA und Proteinen im zentralen Dogma (oder Zentralen Lehrsatz) der Biologie, das sich folgendermaßen darstellen lässt:

Literaturverzeichnis

[As00] Ashburner, M. et. al.: Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. Nature Genetics 25, 2000; S. 25-29. http://www.geneontology.org/cgi-bin/GO/downloadGOGA.pl/gene/association.sgd

[BV00] Brazma A., Vilo J.: Gene Expression Data Analysis. FEBS Letters Volume 480, Issue 1, 2000; S. 17-24. http://eptest.ebi.ac.uk/EP/EPCLUST/

[Ce02] Cherry, J. M.et. al.: Saccharomyces Genome Database, 2002. http://genome-www.stanford.edu/Saccharomyces/ ftp://genome-ftp.stanford.edu/pub/yeast/

[He02] Heymann S. et. al.: Viator - A Tool Family for Graphical Networking and Data View Creation. Very Large Data Bases, HongKong SAR, China, 2002. http://www.dbis.informatik.hu-berlin.de/

[Hy02] Hyun Y.: Walrus - Graph Visualization Tool. 2002. http://www.caida.org/tools/visualization/walrus/

[Ke02A] Keim D. A.: Datenvisualisierung und Data Mining. Datenbank-Spektrum 2, 2002; S. 30-39.

[Ke02B] Kemmeren P. et. al.: Protein Interaction Verification and Functional Annotation by Integrated Analysis of Genome-Scale Data. Molecular Cell 9(5), 2002; S. 1133-1143. http://eptest.ebi.ac.uk/EP/PPI/

[KK01] Kuramochi M., Karypis G.: Frequent Subgraph Discovery. Department of Computer Science/Army HPC Research Center University of Minnesota nr. 02-026, 2001.

[KS02] Kumar A., Snyder M.: Protein complexes take the bait. Nature Vol. 415, 2002; S 123-124.

[Ku02] Kuo W. P. et. al.: Analysis of matched mRNA measurements from two different microarray technologies. Bioinformatics vol. 18, 2002; S. 405-412. http://bioinformatics.oupjournals.org/cgi/content/abstract/18/3/405

[Me97] Mewes, H. et. al.: Overview of the yeast genome. Nature 387, 1997; S. 7-8.

[Me02] Mewes H. W. et. al.: MIPS: a database for genomes and protein sequences. Nucleic Acids Research 30(1), 2002; S. 31-4. http://mips.gsf.de/proj/yeast

[Mo02] MolMineAS, Norway, 2002; J-Express v. 2.1. http://www.molmine.com/

[Mu00] Munzner T.: Interactive Visualization of Large Graphs and Networks, Ph.D. Dissertation, Stanford University, 2000. http://graphics.stanford.edu/papers/munzner/thesis/

[Pe01] Peèr D. et. al.: Inferring Subnetworks from Perturbed Expression Profiles. ISMB, 2001.

[Sp98] Spellman P. T. et al.: Comprehensive Identification of Cell Cycle-regulated Genes of the Yeast Saccharomyces cerevisiae by Microarray Hybridization. Molecular Biology of the Cell 9, 1998; S. 3273-3297.

Comparative Evaluation of Microarray-based Gene Expression Databases

Hong-Hai Do[†], Toralf Kirsten[†], Erhard Rahm[‡]

[†] Interdisciplinary Centre for Bioinformatics, [‡] Department of Computer Science
University of Leipzig
www.izbi.de, dbs.uni-leipzig.de

Abstract. Microarrays make it possible to monitor the expression of thousands of genes in parallel thus generating huge amounts of data. So far, several databases have been developed for managing and analyzing this kind of data but the current state of the art in this field is still early stage. In this paper, we comprehensively analyze the requirements for microarray data management. We consider the various kinds of data involved as well as data preparation, integration and analysis needs. The identified requirements are then used to comparatively evaluate eight existing microarray databases described in the literature. In addition to providing an overview of the current state of the art we identify problems that should be addressed in the future to obtain better solutions for managing and analyzing microarray data.

1 Introduction

With genomes of several organisms, especially the human genome, completely sequenced, the main focus of genomic research has shifted to using these sequences in order to understand how genes and ultimately entire genomes are functioning. Although all cells in an organism carry the same genetic information, only a subset of the genes is active, i.e. expressed, conferring unique properties of the cells in their specific conditions. Analyzing the behavior of the genes, i.e. whether and to what degree they are expressed, can help characterize and understand the functions of genes. In particular, it can be analyzed how the activity level of genes changes under different conditions such as for specific diseases, before and after the use of specific drugs, etc.

Various methods have been developed for detecting and measuring gene expression, including Northern Blotting [AK77], Differential Display [LP92], Reverse Transcription-Polymerase Chain Reaction (RT-PCR) [SW95], EST (Expressed Sequence Tag) Clustering [VE98], Serial Analysis of Gene Expression (SAGE) [VZ95] and microarrays [SS95, LD96, Na99]. Microarrays are quickly becoming the predominant approach because they allow performing expression analysis on a very large scale, i.e. to measure and study the expression of thousands of genes simultaneously. As a consequence, huge amounts of data are produced with every experiment. Moreover, the amount of data being produced is expected to explode with the falling cost of microarray technology.

Recently, several databases have been developed for storing and analyzing microarray data. Some of them have been reviewed in [GL01] with a focus on the databases' analysis capabilities. However, the broader requirements to build, maintain and use such a database in a flexible way are not sufficiently considered. Furthermore, most of the considered databases are not available to the public and/or have not been presented in scientific publications.

To obtain a better overview about the current state of the art in using database technology for gene expression analysis, we review the available microarray databases described in recent scientific publications. For this purpose, we first discuss the major requirements for managing microarray data. We cover important database-related issues that have been left open in [GL01], e.g. performance aspects, data integration, and the coupling of analysis/data mining with the database. We then use these criteria to comparatively evaluate eight database implementations and thus assess the current state of the art. We hope that our requirement analysis and evaluation helps identifying fruitful areas for future research and guiding the design and development of more powerful solutions for microarray data management.

The rest of the paper is organized as follows. In the next section we provide a short introduction to microarray technology. In Section 3 we present the main requirements for a microarray database. Section 4 compares the selected databases according to the identified requirements and criteria. In Section 5 we conclude and point to database-related problem areas for future work.

2 Microarray-based Gene Expression Measurement

The genetic information in the DNA is organized within two complementary strands consisting of sequences of four different nucleotides, Adenine (A), Thymine (T), Guanine (G) and Cytosine (C). Adenine and Guanine are the complements of Thymine and Cytosine, respectively. When two complementary sequences find each other, they hybridize (i.e. bind together). Gene expression is the cellular process that turns the genetic information of the DNA into proteins, which ultimately determine the morphology and functionality of a cell. Protein synthesis starts with the so-called transcription process, in which the genetic information of the DNA is transferred to the short-lived messenger RNA (mRNA). Measuring the mRNA abundance, i.e. *the transcription* or *expression levels*, in various tissues and under different environmental conditions can help understand the dynamic functioning of genes as well as their mutual influence in the regulatory network.

2.1. Microarray Principle

Microarrays are based on the same basic principle: the preferential binding of complementary, single-stranded nucleic-acid sequences. On a microarray (also called a *chip*), known sequences called *probes* are attached at fixed locations (*spots*). There are two variants of the microarray technology:

- *cDNA arrays (spotted arrays)*: This is the oldest microarray technology and was developed at Stanford University. It is based on immobilizing complementary DNA (cDNA) probes of length of 500~5,000 bases (nucleotides), each representing a gene, to a solid surface such as glass using robot spotting.
- *Oligonucleotide arrays*: These arrays use shorter sequences as probes, so-called oligos of 20~80 bases. Unlike in spotted arrays, a gene is represented by a set of oligos, i.e. a *probeset*. This technique was developed by Affymetrix, Inc.

2.2. Experiment Design

Figure 1 gives a schematic overview of a *microarray experiment*. In a cell, when a gene is expressed, mRNA transcripts are produced. In a microarray experiment, these transcripts, also called *targets* in the experiment context, need first to be isolated from the

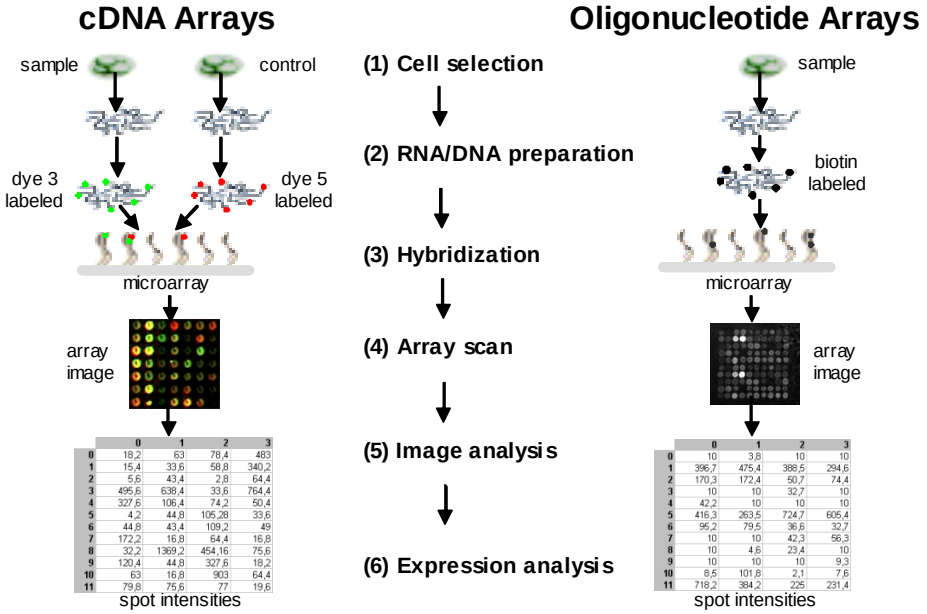


Figure 1. Schematic overview of a microarray experiment

sample of interest (Step 1), reverse-transcribed to cDNA, and tagged with a particular fluorescent dye to mark their origin (Step 2). In case of spotted arrays, transcripts from two samples, e.g. cells from a normal tissue for control and cells from a disease tissue for testing, are needed. When bathing the microarray in the target mixture, the *hybridization* process (Step 3) takes place and the available transcripts bind to the corresponding probes on the chips.

After the hybridization phase, the array is scanned to determine how much of each target is bound to each spot. The *scan* process delivers an image of the array showing spots with different color/brightness intensities depending on how many fluorescent targets are bound to the corresponding probes (Step 4). Finally, these intensities are measured and corrected against the background noise using some *image analysis* software to produce the expression level of each gene (Step 5). For oligonucleotide arrays, the intensities of the spots refer to oligos and are to be combined to produce a single intensity value for the corresponding gene.

To examine gene expression levels under various aspects, e.g. in different tissues or in a time series, an *experiment series* is to be conducted. In general, expression data is of a multidimensional nature: each measured expression level is a point in an *n*-dimensional space with dimensions such as the genes, gene functions, and the different conditions under which the genes have been studied. As we will see later, various analysis approaches can be employed in Step 6 to infer and interpret gene functions from expression data.

3 Database Requirements for Gene Expression Analysis

In this section we discuss the major requirements for microarray databases supporting gene expression analysis. In particular, we consider criteria from the following areas:

- *Data Characteristics*: We analyze which types of data need to be managed and their characteristics to be considered
- *Management of Annotation Data* describing the semantics of the expression data measured in the experiment
- *Data Integration*: In addition to the data generated by the microarray experiment itself, gene expression analysis should exploit annotation information available from public sources. We examine which data is useful and how it can be integrated
- *Data Interfaces* for data exchange (import, export)
- *Access Control* to avoid unauthorized database access in a multi-user environment
- *Data Normalization*: to improve the quality of gene expression measurements, which may suffer from noise due to various experimental fluctuations
- *Data Analysis*: which approaches are useful for gene expression data analysis
- *Tool Integration*: coupling of analysis algorithms and existing tools with the database

In the following we elaborate on these criteria in more detail.

3.1. Data Characteristics

Gene expression analysis requires various kinds of data, which are not only produced directly by the microarray experiment but can also stem from other sources. We distinguish between *Image*, *Expression* and *Annotation Data*. The latter is further divided into *Gene*, *Sample* and *Experiment Annotations*. Table 1 summarizes their characteristics and usage in gene expression analysis.

Data		**Source**	**Type**	**Characteristics**	**Usage**
Image Data		Array scan	Binary	large files	Generation of expression data
Expression Data		Image analysis	Number	fast growing volume	Visualization, statistical and cluster analysis
Annotation Data	Gene	External public sources	Text	regularly updated	Interpreting / Relating / Inferring gene functions
	Sample and Experiment	User input		user-specified, often free text	

Table 1. Relevant types of data and their characteristics

Image Data. Images are produced as large files in the array scan process. They represent the starting point for expression analysis. Because image analysis software may be changed or updated, both the images and their association with the generated expression data should be managed so that the previous analysis results can be reproduced and corrected. Access frequency is relatively rare and mostly read-only. Images may be stored within the database itself or in the file system with the file names or URLs kept in the database.

Expression Data. Expression data, i.e. numbers indicating gene expression levels, represents the core of a microarray database. It is of high volume and fast growing. Gene expression levels computed by different technologies, such as cDNA arrays, oligonucleotide arrays, as well as other non-array technologies like SAGE possess different semantics, and therefore are difficult to compare with each other without being normalized first. Unlike images, expression data is accessed more frequently. Typically, analysis poses high performance requirements due to the high data volume and the frequent need of interactive analysis requiring short response times. This asks for the use of advanced DBMS techniques such as materialized views, indexing and parallel processing that may have to be tailored to specific analysis needs.

Annotation Data. Annotations are metadata describing the expression levels measured in a microarray experiment, often in the form of textual descriptions. They help the user

in interpreting the detected gene expression levels, especially for inferring and relating gene functions. We distinguish between the following kinds of annotation data:

- *Gene Annotations*: Sequences placed on microarrays usually represent already known genes. Their annotations, e.g. names, currently known functions, location on chromosome, etc. are essential for interpreting the measured expression levels. Such information has been continuously collected, regularly updated and made available in various public data sources.
- *Sample Annotations*: This data describes how the targets have been extracted and prepared for hybridization. Moreover, it also includes biological descriptions, such as the source and characteristics of the sample, e.g. tissue and disease, any genetic and chemical manipulation and stimulation, any *in vivo* or *in vitro* treatments applied. For patient-related measurements personal characteristics such as age, sex and clinical status information can provide further important criteria for analysis.
- *Experiment Annotations*: This data describes primarily the technical process of the experiment. In particular it captures the protocols and parameter settings used by the machine and software for hybridization, for washing and scanning the array.

Typically gene annotation data has to be integrated from external public sources, while sample and experiment annotations need to be manually specified by the user for every new experiment. This leads to special requirements that are discussed in the following.

3.2. Management of Annotation Data

Because annotation data comes from heterogeneous sources, such as external databases and user input, it is essential to capture and organize annotation data in a uniform and flexible way so that it can be effectively used in analyzing expression data. Current databases often use free-text fields to capture annotation data, leading to two problems. First, because different sources and users often use different vocabularies, free-text fields tend to introduce large annotation discrepancies. Second, each free-text field potentially contains many terms or values, making them difficult to be queried in the database. Heterogeneous annotations make it difficult to identify comparable experiment results and to perform cross-experiment analysis.

As a result free-text fields should largely be avoided for annotations. Rather, annotations should be split into atomic items or categories with clearly defined semantics of simple data types, such as numbers of predefined units or values from a predefined list. The items as well as their values should be specified using a controlled *vocabulary*, which can either be specifically developed for local use only, i.e. a local vocabulary, or based on an existing standard, i.e. a standardized vocabulary. Moreover, the categories should not only be collected in flat vocabularies, but also organized into multiple levels, e.g. *taxonomies* and *ontologies*, to increase their expressiveness and support more focused analysis capabilities. A taxonomy, such as the gene function taxonomy of the GeneOntology (GO) Consortium [GO00], is a specialization/generalization hierarchy of categories, which are connected with each other by *is-a* relationships. Ontologies such as TAMBIS [BG99] often represent additional semantics, e.g. complex networks of categories.

The database representation for annotations should take into account that the relevant items/categories and vocabularies change over time, e.g. if the experiments change their biological focus. Figure 2 illustrates two representation schemes for annotation data, a straightforward relational approach and the so-called *Entity-Attribute-Value* (EAV) ap-

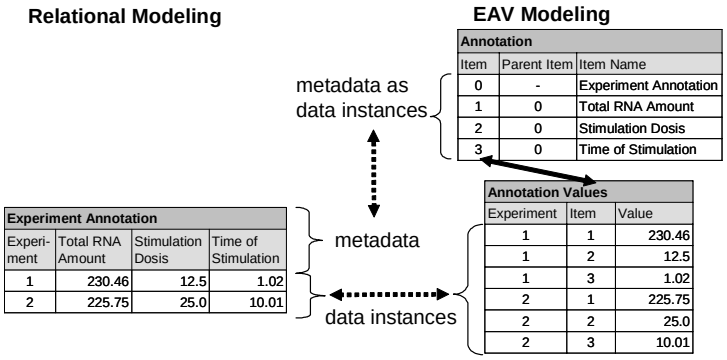


Figure 2. Relational vs. EAV approach for modeling annotation data

proach [NB98]. In the former method, annotation categories are modeled as attributes of tables and the captured values are stored as instances. This simple approach makes it easy to control the correctness of annotation values and to use them for data analysis. However, it is only suitable for annotations that rarely change over time because the database schema has to be modified whenever different annotations are needed. Moreover, hierarchically organized categories need additional support.

On the other side, the EAV method is more flexible and robust against changes in annotation categories. Here, each annotation category or item appears as a uniquely identified instance in a metadata table, the *Annotation* table, which can be easily extended to capture more items. Furthermore, the items can also easily be organized in a hierarchical way. The captured values for each item are stored in another table, the *Annotation Value* table. The price for the flexibility of the EAV approach is that queries are hard to formulate and that the two tables always need to be joined to associate the items with their values. This drawback however may be reduced by building materialized views to simplify and speed up frequent queries involving annotation data.

3.3. Data Integration

We first discuss the challenges directly resulting from the information need of gene expression analysis and then the mechanisms for integrating gene annotation data.

Integration Requirements. Gene annotation data is stored in various public data sources accessible on the web. For instance, gene sequences placed on an array often stem from a sequence database, such as GenBank [WC02], which maintains all known nucleotide and protein sequences with different annotations, e.g. on bibliographic references, organisms, coding regions, repeat regions, mutations, etc. To characterize the functions of genes, their expression patterns should be related with the functions of their products, e.g. proteins. SwissProt [BA00] is a curated protein sequence database providing for each protein sequence extensive annotation information, such as functional descriptions, structures, associated diseases etc. Moreover, the genes can be examined in a broader context, namely in the network of interactions between their proteins. This information can be exploited from KEGG [KG00], a collection of pathway maps computerizing the network information of molecular interactions, such as metabolism, signal transduction, cell cycle, etc.

Since a gene is often represented by multiple sequences in GenBank, annotations at the sequence level are often impractical for functional gene analysis. Hence a major requirement is to integrate sequence-level annotations from different sources to provide

gene-oriented annotations, e.g. for gene expression analysis. This is already supported by several databases. LocusLink and RefSeq [PM01] are the sources of choice for curated annotations and sequences of known genes. Other databases such as UniGene [WC02], TIGR [QC01] and Ensembl [HB02] have been constructed using different automatic gene prediction algorithms and provide computed annotations for the predicted genes. The Human Genome Browser (HGB) [KS02] maps the sequence of the genes maintained by different databases/predicted by different algorithms uniformly onto the genome, providing a powerful visual mean for comparing genes from different databases as well as for relating the genes with other annotations, like tandem repeats, CpG islands, homology between species, which can also be mapped onto the genome. Finally, microarray vendors also provide annotations for the genes on their own chips. For example, Affymetrix users can exploit the NetAffx database for probeset annotations for all Affymetrix chips [LL02]. The annotations include information integrated from LocusLink, UniGene and SwissProt, as well as various in-house computed annotations.

Typically, each database uses proprietary gene identifiers so that the same gene may be found under different identifiers in different sources. Furthermore, vendor-proprietary gene identifiers such as Affymetrix probesets are unknown in public annotation databases and not suitable for referencing in scientific publications. Therefore, an essential requirement in integrating gene annotation data is to relate the corresponding genes between public annotation databases with the proprietary genes of microarray vendors.

Integration Mechanisms. Traditional approaches for data integration are *Virtual* and *Materialized Integration*. In the former approach the data is retrieved from the corresponding sources when it is needed, while the latter locally replicates the data from the external sources. We further differentiate between two variants of the virtual approach, namely *Web Link* and *Federated* integration.

- *Web Link Integration*: This approach is followed by most current databases and only stores the *accession keys*, the unique keys to access data entries in the external sources. Using accession keys, web links can be built automatically, allowing the user to navigate to the corresponding source in order to obtain annotation information for the genes of interest. While requiring only little integration effort, this approach shows significant limitations. Firstly, it is not possible to consider several genes, for example in an identified gene cluster, at the same time. Secondly, and more importantly, it is not possible to directly relate the annotations and expression of genes for database queries or data mining.
- *Federated Integration*: In this variant of virtual integration, the schemas of the relevant sources have first to be integrated to a global schema. Determining a consistent global schema is a major problem due to typically large degrees of semantic heterogeneity between different sources, despite the availability of some global taxonomies (GO etc.). Furthermore, a complex mediator software is needed supporting queries against the global schema by executing relevant subqueries at the respective data sources and combining their results. The approach also suffers from the only rudimentary query capabilities of public sources, typically based on string/pattern matching of text. Furthermore, strategies to automatically deal with possibly dirty and overlapping data between different sources have to be developed and incorporated in the query processing engine. Moreover, query processing depends on the availability and performance of the corresponding sources. On the positive side, the data itself needs not be replicated and the most up-to-date data can be retrieved and analyzed.

- *Materialized Integration*: This approach corresponds to data warehousing and requires extensive preprocessing effort. Not only the source schemas have to be integrated, but the data has also to be extracted from the single sources, transformed, cleaned, and then uniformly stored in the microarray database (warehouse), together with expression measurement data. As external sources are regularly updated, automatic techniques are needed to refresh the local data on a continuous basis. Once the data has been integrated, the warehouse approach promises significant advantages because all relevant data is directly accessible for analysis. This can help to provide both good performance and extensive analysis capabilities.

3.4. Data Interfaces

We first discuss data exchange interfaces w.r.t. other databases and software tools. We also discuss security aspects, i.e. how to control data access of users.

Data Exchange. Because experiments may be continuously conducted by different users in different labs, a public database should provide the users with possibilities to import, i.e. submit, their data for management and analysis. Furthermore, some users may want to export the data so that they can use external tools or programs for data analysis. Hence, interfaces for both import and export are required.

The most common way for data exchange is to support a particular flat file format. Expression data can be easily organized in a matrix, the gene expression matrix [BH01], with rows representing genes and columns the investigated samples. Hence, tab-delimited files represent a straightforward way to exchange expression data. This file format has an essential disadvantage that it does not include the corresponding experiment and sample annotations. In contrast, annotation can be easily specified using XML which has already been widely used to exchange data over the web. Several efforts have developed proposals for a standard XML format for microarray data, e.g. MAGE-ML[1], GEML[2], and GeneXML[3].

Access Control. The control of user access to expression data is an important criterion for the acceptance of a microarray database due to two main reasons. First, the experimenter usually wants to hold back his/her expression data and analysis results until they have been published in some journal or conference contribution. Second, annotation data may contain sensitive person-related information, such as patient and clinical data.

As usual, access control has to consider the individual users, the available data as well as the access rights or functions. This may be achieved with the authorization concept of the underlying DBMS or by a specific implementation. With respect to the *users*, the database should provide some mechanisms for building a hierarchy of individual users, groups, and roles. Regarding the *data*, different levels of granularity should be distinguished, such as expression data of an experiment/experiment series or annotation data. Finally, different access *functions* should be supported such as data import, export, or performing certain analysis types.

3.5. Data Normalization

Raw expression data produced by the image analysis process still contains noise. In particular, each step in target and probe preparation, in the hybridization, wash and scan

[1] http://www.mged.org/Workgroups/MAGE/mage-ml.html
[2] http://www.rosettabio.com/products/conductor/geml/default.htm
[3] http://www.ncgr.org/genex/genexml.html

process represents a source of fluctuations which can influence the determination of expression data in different ways. For example, the efficiency of the hybridization reaction depends on a number of experimental parameters, such as temperature, time, and the overall amount of available mRNA. Since the reliability of expression patterns derived from array data is essential for their interpretation, a data normalization step aiming at reducing the effects of such fluctuations is necessary.

Currently, many strategies have been proposed for normalizing data from a single experiment or from an entire experiment series. Descriptions and evaluations of the various strategies for cDNA and oligonucleotide arrays can be found in [SB00] and [HB01], respectively. For a single experiment, common normalization strategies perform a division by a constant approximately determined by average intensity either of all spots on the array (ratio vs. total) or of a few control genes (ratio vs. control), such as the so-called housekeeping and spiked genes, whose expression behavior is already known or predictable. For multiple experiments, one approach is to normalize them against one reference experiment which has been conducted for a control sample. After being adjusting to a common standard, the results from different experiments can be compared.

Because there is still no agreed-upon standard procedure, the common normalization strategies should be provided so that the user can choose to pre-process his/her expression data. Moreover, not only the normalized, but also the raw expression data should be stored in the database, so that a re-normalization can be performed later on, e.g. for testing and evaluating novel normalization strategies.

3.6. Data Analysis

The analysis process takes the normalized expression data and tries to derive the relationships between the genes and samples. Most methods for gene expression analysis have already been developed and used in other areas, especially data warehousing, data mining, and statistics. We differentiate between the following families of analysis approaches:

Querying/Reporting. This standard database access allows the user to navigate in the database and to retrieve a subset data of interest for further study or visualization. To simplify the construction of frequent queries and speed up their execution, canned queries and reports should be supported. They are pre-defined database queries, which are stored so that they can be executed at any time with different user-specified parameter values. For example, canned queries can be defined to filter genes based on specific expression level thresholds and/or functional annotation.

Online Analytical Processing (OLAP). OLAP has been widely used in data warehousing to analyze multi-dimensional data; recently, the use of OLAP has also been proposed for the biotech domain [Hu01]. Because of the multidimensionality of gene expression data, this technique represents a promising approach to gene expression analysis. Assuming a proper representation for dimensional annotations, the user may interactively navigate through different levels in the hierarchy of a dimension, such as the GO function hierarchy of genes, to obtain and compare summarized information about gene expression patterns.

Data Mining. Data mining supports the detection of interesting patterns in large data sets and has commonly been used for analyzing expression data. There are unsupervised approaches, e.g. clustering, as well as supervised schemes, e.g. classification methods.

- *Clustering*: represents the most common analysis method for expression data. The goal of clustering is to group together objects, i.e. gene or samples, with similar properties. So far, many algorithms, such as hierarchical, K-mean clustering algorithms, and Self-Organizing-Maps (SOM) [BR02], have been developed and successfully employed to analyze expression data. Typically, genes are clustered to identify co-regulated and functionally related genes. Furthermore, clustering can also be performed for samples. Samples with similar expression patterns may constitute some new, previously undefined subgroups, e.g. for diseases like tumors. These findings can be useful for designing treatment procedures for different groups of patients.
  Clustering is often accompanied by dimension reduction methods, which can either identify and disregard the less informative dimensions or establish a new smaller set of dimensions as combination of the original dimensions. These methods include Multidimensional Scaling (MDS), Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) [Do02].
- *Classification*: This supervised approach is often based on machine learning and aims at assigning predefined classes of known characteristics and functions to given expression patterns. Popular classification methods include linear discriminants, decision trees, and Support Vector Machines (SVM) [BG00]. Typically, such classifiers are first trained on a subset of data, for which classification is already known, and then tested to find classification for another subset of data.

Statistics. Statistical methods are emerging to account for multiple sources of variation when trying to pool information from many microarrays and to identify genes exhibiting significant differential expression. The ANOVA approach [KM00] decomposes the appropriately transformed expression measurement as a linear combination of effects from different sources of variation. Also in this context, several other statistical techniques have also been employed, including the permutation-testing and p-value adjustment [DY02], the t-test and Wilcoxon test [Pa02].

Visualization. All analysis methods need to summarize the results in a comprehendible way for human interpretation. Using different techniques, such as scatter plots, dendrograms, charts and graphs, a large amount of data can be surveyed and examined simultaneously. In particular, visualization is needed to display the results of clustering.

3.7. Tool Integration

Typically, data analysis has to be performed iteratively and in interaction with the user. This requires a close integration of the analysis methods with the database. For the various types of analysis discussed above, many algorithms and visualization support are already available in powerful software tools. These tools should be usable together with the microarray database which also helps to limit development effort. We distinguish between three forms of tool integration.

Loose Integration. In this scenario, only little integration effort is needed. The user uses the export interface of the database to export a subset of data of interest, typically gene expression levels, to a flat file, which is then imported in the corresponding tools. Its main drawback however consists in the lack of annotation data in those tools for interpreting the expression patterns. Even so, for many proprietary tools it represents the only way to analyze expression data.

Transparent Integration. This approach can be employed to integrate tools which provide some API to their functionalities. A single user interface can be built covering

multiple tools addressing different steps in gene expression analysis. The communication between the tools and the database can be based either on direct database queries or flat file export and import, which is however hidden from the user.

Tight Integration. As opposed to loose integration, this approach requires analysis algorithms to work directly on the database. It represents the most promising integration form because it can exploit all available data in the database and achieve the best performance. However, it implies a high implementation effort to re-develop the approaches as new database applications and to tune the database, or to directly integrate the analysis approaches into the DBMS, e.g. as stored procedures or special type extensions.

4 System Evaluation

According to the introduced criteria and requirements we compare in this section several public microarray databases, which we could identify from recent scientific literature. Being presented and discussed in scientific publications, their approach has actually been approved by the research community and should show impact on future work. However, we have encountered a number of systems, such as GEO [ED02], Gene Expression Atlas [SC02], HugeIndex [HW02], yMGV [CD02], READ [BK02], and SGD [BJ01], which are still at early stage of development and/or have not been described with sufficient detail for our purpose. Therefore, we do not consider those databases but only focus on 8 databases listed in Table 2. In addition to the information from the publications, we also test and consider the functionalities provided by the websites of the corresponding databases.

Databases	Organization	References
ArrayDB	National Human Genome Research Institute (NHGRI) *http://genome.nggri.nih.gov/arraydb*	[ER98]
ExpressDB	Harvard University *http://arep.med.harvard.edu/ExpressDB*	[AR00]
GeneX	National Center for Genome Resources (NCGR) *http://genebox.ncgr.org/genex*	[MC01]
GIMS	University of Manchester *http://www.cs.man.ac.uk/~norm/gims*	[CP01, PK00]
M-CHIPS	German Cancer Research Centre (DKFZ) *http://www.mchips.de*	[FH02, FH01]
RAD2	University of Pennsylvania *http://www.cbil.upenn.edu/RAD2*	[SP01]
SMD	Stanford University *http://genome-www4.stanford.edu/MicroArray/SMD*	[SH01]
YMD	Yale University *http://info.med.yale.edu/microarray*	[CW02]

Table 2. Microarray databases in the evaluation

4.1. Technical Implementation

Table 3 shows an overview of the databases according to their technical implementation. While all databases are accessible over the internet, they are intended to store and support analysis of microarray data generated by local labs. However, external users can pose queries to the data, which has been made available to the public. Only very few projects, in particular, ArrayDB, ExpressDB, GeneX and SMD, are open-source, allowing a local installation. All databases in our evaluation make use of DBMS technology, which is in most cases a commercial relational DBMS. GIMS represents an exception by using POET, an object oriented DBMS. All databases, except for GIMS, provide web interfaces for data access, which have been implemented using common web technologies, such as

Perl and Javascript. For data analysis routines, some databases also use special programming environments, such as the statistic language R and the scientific software package MatLab.

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
Access	no public data submission but public queries							
Open source	Yes			No			Yes	No
DBMS	Sybase	Sybase	PostgreSQL, Sybase	POET	Post-greSQL	Oracle	Oracle	Oracle
Program languages	Perl, Java	Perl, Javascript	Perl, Java, R	Java	C, Perl, MatLab	Perl, Java	Perl, C	Perl
GUI	Web	Web	Web	Java	Web	Web	Web	Web

Table 3. Technical implementation of the databases

4.2. Supported Kinds of Data

Table 4 shows the different types of data managed by each database. Only few databases, in particular ArrayDB, SMD, and YMD, consider storing array images for later reference and analysis. The images are not managed by the DBMS, but in file systems. The DBMS only stores the path to the image files, which is to be specified explicitly.

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
Images	in file system	no	no	no	no	no	in file system	in file system
Expression Data	cDNA	cDNA, Oligo, SAGE	cDNA, Oligo, SAGE	cDNA (public data sets from SMD)	cDNA, Oligo, SAGE	cDNA, Oligo, SAGE	cDNA	cDNA, Oligo

Table 4. Images and expression data managed by different databases

cDNA arrays are supported by all databases. Several databases are able to store expression data produced by other technologies. The new emergence of the oligonucleotide arrays has already been taken into account by several solutions. In particular, expression data from Affymetrix arrays can be managed by ExpressDB, GeneX, RAD2 and YMD. Furthermore, a few databases including ExpressDB, GeneX, and RAD2 also support SAGE expression data. Currently, SMD represents the biggest microarray database with more than 538 million expression data points from 25 thousands experiments (as of July 2002). However, despite the huge data amount and the requirement for query performance, no experience has been reported so far concerning the use of advanced DMBS techniques, such as materialized views and parallel processing.

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
Sample Ann.	tissue, cell type	1 free-text field	sex, age, tissue, dev. stage, ...	no	very comprehensive lists of annotation information	sex, age, disease, dev. stage, ...	sex, age, status, ethnicity, ...	no
Experiment Ann.	array printing, environmental conditions	1 free-text field	hardware and software parameters	no		RNA amplification, labeling protocol, scan parameters	yes	no

Table 5. Sample and experiment annotations

Because gene annotation represents data to be integrated from external sources, we will discuss it later in the next section together with the data integration mechanisms. Table 5 shows the information which currently can be specified for sample and experiment annotation. Most databases allow the specification of some information, such as sex, age, tissue, developmental stage, to annotate the sample being examined as well as the protocols, hardware and software parameters used to conduct the experiment. However, the degree of details varies drastically from database to database. Some databases,

such as GIMS and YMD, completely lack the possibility to specify sample and experiment annotations.

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
Capturing	no controlled vocabularies	no controlled vocabularies	local vocabularies	-	local vocabularies	standard vocabularies	local vocabularies	-
Modeling	relational	relational	relational	-	EAV	relational	relational + EAV	-

Table 6. Capturing and modeling/storing sample and experiment annotation

Table 6 shows the techniques used in current databases for capturing and modeling sample and experiment annotation. Typically, free text fields are provided to specify their own descriptions. Especially, ExpressDB provides a single text field for completely annotating the sample and the execution of the experiment, respectively. In contrast, M-CHIPS does not allow any free text fields and employs comprehensive lists of strictly defined annotation items, enforcing the user to specify very detailed information about the sample and the experiment. Because of the pre-definition of all annotation items and their values, annotation data in M-CHIPS is uniform across different experiments and can be exploited for statistical analysis. Other databases try to limit the negative effects of free text by enforcing controlled vocabularies when applicable. While GeneX and SMD use vocabularies developed by local users, RAD2 exploits standard vocabularies employed in other sources, in particular NCBI Taxonomy, MGD mouse anatomy and KEGG disease table.

Typically, the databases use a standard representation with fixed attributes for sample and experiment annotations. Only M-CHIPS and SMD follow the EAV approach and hence are more flexible in case new annotation information is to be captured. While M-CHIPS strictly applies the approach to store its annotation data, SMD employs both approaches and only uses EAV when necessary.

4.3. Data Integration

Table 7 shows the public data sources, which have been integrated with the current microarray databases. We observe that web linkage represents the most commonly used integration mechanism. Despite the limitation that the annotation data residing in the external source cannot be programmatically involved to analyze the local expression data, this approach requires almost no integration effort and is the fastest way to make a database solution ready for public use. Among others, UniGene, GenBank, SwissProt and KEGG represent the mostly referenced sources.

On the other side, the federated approach of virtual integration has not been exploited by any current databases. Similarly, the materialized approach also has found only very little use. In particular, still very limited gene annotation data has been integrated and

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
Web Link	UniGene, dbEst, GenBank, KEGG	BIGED	SGD, MGD, dbEST, GenBank, KEGG, SwissProt	no	yes	GenBank, AllGenes, KEGG	dbEST, GeneMap, LocusLink, SwissProt,	UniGene DRAGON, SOURCE
Federated	no							
Materialized	no	names, functional groups for yeast (MIPS)	no	functional groups for yeast (MIPS)	GO functions	no	GO functions (SGD), gene names (WormPD), UniGene	no
Auto. Update	-	no	-	no	no	-	yes	-

Table 7. Gene annotation data integrated from external public databases

replicated in the current microarray databases. Mostly, the data has been imported just once, as for example in ExpressDB and GIMS, and no mechanism for continuously updating them is provided. So far, SMD represents the single effort to comprehensively integrate gene annotation data from different sources while providing mechanisms to automatically keep the local data updated with the sources.

4.4. Data Interfaces

In this section we examine how interfaces for data exchange and user access have been implemented by the current databases.

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
Import	tab-delimited	tab-delimited	tab-delimited, GeneXML	tab-delimited	tab-delimited	tab-delimited	tab-delimited, Genepix, Scanalyze	tab-delimited, Genepix, GPMerge
Export		no		no	no	no	tab-delimited, TreeView	tab-delimited, Excel, CLUSTER
Automation	directory scan	no	no	no	no	no	batch import	batch import

Table 8. Data exchange interfaces and mechanisms

Data Exchange. Table 8 shows the interfaces provided by the corresponding databases for data exchange. Data exchange mostly addresses gene expression levels, so the ASCII tab-delimited file format is widely supported for both import and export.

SMD and YMD allow direct import of data from proprietary files produced by particular image analysis software such as Genepix and Scanalyze. Furthermore, they also support exporting data to the formats required by some analysis tools, such as TreeView, CLUSTER and Excel. GeneX represents the first effort so far to use XML as exchange format. In particular, it allows microarray data, i.e. both annotation and expression data, to be imported and exported using the proprietary format GeneXML.

Mostly the user has to manually initiate the import and export process from the database website. Automation for import is provided only by ArrayDB which can automatically scan a specified directory for import files. SMD and YMD also support import of multiple files which however have to be specified first on the database website.

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
User/group	application-based	application-based	application-based	-	separate databases for each group	application-based	application-based	application-based
Granularity	experiment		experiment	-	experiment series	experiment	experiment	experiment series
Function	No							

Table 9. Control of data access

Access Control. Table 9 shows how the data access is implemented in current databases. Typically, the databases implement their own user and group hierarchies (at application level) and do not make use of the DBMS-provided user/group concept. M-CHIPS represents an exception by providing each group of users, which perform similar experiments, with a separate logical database.

Typically, the finest granularity of data that can be assigned to the user is the experiment, which consists of both annotation and expression data. A distinction between these two kinds of data for access control is not yet implemented in any database. A few databases, such as YMD, assign an entire series of related experiments to a user. This is also the case with M-CHIPS, as a group-specific database in M-CHIPS also represents a series

of related experiments. Finally, no database supports the restriction of functions that can be performed by a particular user on the assigned data set.

4.5. Data Normalization

Table 10 shows the normalization strategies supported by the different databases. Several databases, such as ArrayDB, ExpressDB and GIMS, do not implement any normalization strategies, leaving to the user the task to normalize the data before uploading it. Hence, the user has to be aware about whether the data stored in the database has been normalized, and if so, using which strategy.

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
Normalization methods	no	no	average, ratio vs. control	no	robust affine-linear regression vs. control	ratio vs. total, ratio vs. control	2 strategies with scaling factors	no
Expression data	-	-	raw + normalized	-	raw + normalized	raw + normalized	raw + normalized	-

Table 10. Supported normalization strategies

On the other side, a few databases, e.g. SMD, M-CHIPS and RAD2, provide integrated strategies and allow the user to choose the strategy to normalize the data being uploaded, although the strategies, apparently tailored to the characteristics of the local expression data, are very different between the databases. They also store both the raw and normalized expression data, allowing a re-normalization using another strategy.

4.6. Data Analysis

We now examine the facilities provided by the databases for data analysis, in particular Querying/Reporting, Data Mining and Statistics and finally, Visualization. So far, no database makes use of OLAP technologies. We also indicate how the analysis tools have been integrated with the database.

Querying and Reporting. Table 11 shows the querying and reporting facilities provided by the single databases. All databases support a query tool, mostly web-based, which all operate directly on the DBMS (i.e. tight integration). The common approach, as followed by various databases, such as ExpressDB, SMD, YMD, and RAD2, is first to allow the user to select or search for the experiments of interest, and then to filter the relevant genes by specifying search criteria for the thresholds, the intervals of expression values or gene annotation information, such as name, organism, and disease. In contrast to the simple HTML-based in other databases, ArrayDB provides a comprehensive, integrated graphical tool with more interaction options for user queries.

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
Software Tools	ArrayViewer, MultiExperimentViewer (Web)	Web	Web	Java	Web	Web	Web	Web
Integration	tight integration							
Functionalities	selecting, filtering experiments, filtering genes	selecting experiments, filtering genes	selecting, filtering experiments	canned queries	filtering genes	selecting experiments, filtering genes, canned queries	selecting experiments, filtering genes	selecting experiments, filtering genes

Table 11. Querying and reporting facilities

GIMS and RAD2 allow defining and storing canned queries to answer frequently asked questions. For example, GIMS provides queries for detecting relationships of gene ex-

pression to the gene structure (distribution of introns and exons), to the location of the gene products in the cell, and to the location of the genes on chromosome.

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
Software Tools	no	proprietary	RClust, Eisen, CyberT (Web)	no	proprietary	no	XCluster (Web)	no
Integration		loose	transparent		tight		transparent	
Data mining	no	condition clustering using Pearson correlation	hierarchical, K-means, permutation-based, PCA	no	correspondence analysis, hierarchical clustering	no	hierarchical, K-means, SOM, SVD	no
Statistics	no	no	t-tests, Bonferonni correction, Bayesian variance estimation	no	no	no	no	no

Table 12. Implemented data mining and statistical methods

Data Mining and Statistics. Table 12 shows the data mining and statistical methods currently implemented in the different databases. We can observe that GeneX, SMD and M-CHIPS offer the most comprehensive facilities for data mining, allowing the user to perform various clustering methods, such as the hierarchical and K-means algorithms. While for M-CHIPS, dedicated analysis tools have been developed to operate directly on the database, i.e. tightly integrated, GeneX and SMD transparently integrate the existing clustering tools under their web interface. The user can first identify a data set of interest using the query tool and then immediately specify a data mining method to analyze the data set. The data set is automatically extracted to a file and fed to the data mining tool. ExpressDB also includes a clustering tool, which is, however in contrast to those in GeneX and SMD, only loosely integrated. The data has first to be manually exported from the database, transformed and then imported to the tool for analysis. Similar to ExpressDB, ArrayDB, GIMS and YMD do not have any integrated clustering algorithms. Unlike data mining, integrated statistical analysis has not been supported widely yet. Only GeneX has an integrated tool, CyberT, for performing different statistical tests, such as t-tests, on expression data.

	ArrayDB	ExpressDB	GeneX	GIMS	M-CHIPS	RAD2	SMD	YMD
Software Tools	ArrayViewer, MultiExperiment-Viewer (Web)	MS Excel	RClust, Eisen (Web)	proprietary (Java)	proprietary	no	XCluster, TreeView (Web)	no
Integration	tight	loose	transparent	tight	tight		transparent	
Visualization	zoomable spot map, intensity graph	cluster image	clickable dendrograms, cluster trees	graphical browsers for protein interaction	correspondence analysis biplot	-	zoomable spot map, clickable cluster maps	-

Table 13. Visualization features

Visualization. In Table 13 we present the most remarkable visualization features of the databases. Typically, the clustering tools, which are only integrated in GeneX, SMD, and M-CHIPS, also possess the functionality to visualize their results, the cluster maps. In GeneX and SMD, the maps are clickable so that the user can directly navigate from the cluster result to the genes of interest and their annotation. ExpressDB uses MS Excel to offline visualize the clustering results. GIMS provides a Java-based user interface for browsing and navigating along the protein-protein interaction network. Only ArrayDB and SMD integrate array image with gene expression analysis. Here the user can zoom to individual spots to verify intensity values and obtain other spot-related metadata.

Databases	Advantages	Drawbacks
ArrayDB	• comprehensive graphical query tool	• cDNA array-specific expression data • no local gene annotations • no integrated clustering and statistics
ExpressDB	-	• limited sample and experiment annotation • no integrated data analysis
GeneX	• transparently integrated analysis functionalities (clustering and statistical) • XML (GeneXML) as exchange format	• no local gene annotations
GIMS	• comprehensive library of canned queries	• no integrated clustering and statistics
M-CHIPS	• enforcing local controlled vocabularies in capturing user-specified annotation data • EAV-based management of sample and experiment annotation data	• no local gene annotations
RAD2	• integration of various standard vocabularies for sample annotations • canned queries	• no local gene annotations • no integrated clustering and statistics
SMD	• transparently integrated cluster analysis • materialized integration of gene annotation data and update automation	• cDNA array-specific expression data
YMD	-	• no local gene annotations • no integrated clustering and statistics

Table 14. Main advantages and limitations

4.7. Comparative Discussion

Table 14 summarizes the major advantages and drawbacks we have observed for the various systems. Most databases are able to manage expression data generated using different technologies. Exceptions are ArrayDB and SMD, which focus on cDNA microarray technology. The use of annotation data varies drastically from database to database. For sample and experiment annotation, mostly free-text fields are provided. ExpressDB, for example, uses a single description field for capturing sample and experiment annotation, respectively. A few databases however try to enforce controlled vocabularies (M-CHIPS for locally developed vocabularies to specify sample and experiment annotations, RAD2 to integrate and use standard vocabularies). Usually, a standard relational approach is employed to represent annotation data. The more flexible EAV approach is only supported by M-CHIPS and SMD. SMD integrates and locally replicates gene annotation data from some public sources. Other databases, such as GeneX, M-CHIPS, RAD2 and YMD, provide links to external sources but do not locally store gene annotation data.

The common exchange format for microarray data is the tab-delimited file. So far GeneX represents the only effort to employ XML for both data import and export, which allows to exchange both expression and annotation data. Data access is typically controlled at the experiment level. So far, no distinction has been made between annotation and expression data for access control.

Finally, the databases widely differ in their data analysis facilities. ArrayDB provides a comprehensive graphical query tool for interactive investigation of the gene expression, while other databases offer rather simple web-based query forms. Only GIMS and RAD2 support canned queries. Most databases, in particular ArrayDB, ExpressDB, GIMS, RAD2 and YMD, do not yet support clustering and statistical analysis methods. In contrast, GeneX and SMD exhibit comprehensive facilities for data analysis. In these databases, the different tools are transparently integrated under a uniform user interface, providing a relatively convenient and powerful analysis framework.

5 Conclusions and Future Work

Recently, microarray technology has emerged as a revolutionary technique in molecular biology, allowing to study the expression levels of thousands of genes simultaneously. This massive parallelism has led to an explosion of valuable data to be managed and analyzed. So far, several databases have been developed for microarray data but the current state of the art in this field is still early stage. In this paper, we comprehensively analyzed the requirements for microarray data management. We considered the various kinds of data involved as well as data preparation, integration and analysis needs. The identified requirements are then used to comparatively evaluate eight existing microarray databases described in the literature.

Based on the observed strengths and weaknesses of the current databases we can identify the following major problem areas to be addressed in future research:

- *Data Integration*: Web link-based data integration, as usually implemented in current databases, is not sufficient to support gene expression analysis. Advanced approaches such as federated or materialized integration promising more comprehensive analysis of microarray data, have not yet been investigated in this context. Their implementation poses significant challenges w.r.t. schema and data integration, schema matching, and data cleaning. While techniques from other data integration areas are likely to be useful, the specifics of the bioinformatics domain need to be considered for viable solutions, e.g. to deal with the characteristics of the public sources, such as limited query capabilities, overlapping data, use of different vocabularies etc.
- *Data Analysis*: Current databases only provide simple analysis approaches and do not sufficiently exploit annotation data thereby making only suboptimal use of the obtained expression data. For instance, the multidimensionality of expression data and the typically hierarchical nature of (annotation) dimensions have largely been ignored so far. Hence, the applicability of OLAP technologies for interactive analysis, for which various powerful tools are already available, needs to be explored. This necessitates expression and annotation data be clearly defined and modeled, which also requires further research. Moreover there is a large spectrum of data mining approaches but yet there is no systematic evaluation of their relative strengths and weaknesses w.r.t. gene expression analysis.
- *Performance Optimization*: To achieve high performance for interactive (OLAP) queries and data mining on large amounts of expression data, the use of advanced DBMS technologies such as materialized views, parallel processing, and indexing is to be evaluated. Especially for interactive data mining purposes, new approaches in these areas are likely necessary to achieve short response time.

At the University of Leipzig we have started a project to build a microarray data warehouse for local user groups that aims at taking the discussed requirements into account.

Acknowledgements
We thank the anonymous reviewers for their helpful comments. This work is supported by DFG grant BIZ 6/1-1.

References

[AK77] Alwine, J.C., D.J. Kemp et al: Method for Detection of specific RNAs in Agarose Gels by Transfer to Diazobenzyloxymethyl-paper and Hybridization with DNA Probes. Proc. National

Academy of Science 74, 1977

[AR00] Ach, J., G.M. Rindone: Systematic Management and Analysis of Yeast Gene Expression Data. Genome Research 10, 2000

[BA00] Bairoch, A., R. Apweiler: The SwissProt Protein Database and its Supplement TrEmbl in 2000. Nucleic Acids Research 28(1), 2000

[BF99] Beißbart, T., K. Fellenberg et al: Processing and Quality Control of DNA Array Hybridization Data. Bioinformatics 16:11, 2000

[BG99] Baker, P.G., C.A. Goble et al: An Ontology for Bioinformatics Applications. Bioinformatics 15(6), 1999

[BG00] Brown, M.S., W.N. Grundy et al: Knowledge-based analysis of microarray gene expression data by using support vector machines. Proc. National Academy of Science 97, 2000

[BH01] Brazma, A., P. Hingamp et al: Minimum Information about a Microarray Experiment (MIAME) – Toward Standards for Microarray Data. Nature Genetics 19, 2001

[BJ01] Ball, C.A., H. Jin et al: Saccharomyces Genome Database Provides Tools to Survey Gene Expression and Functional Analysis Data. Nucleic Acids Research 29(1), 2001

[BK02] Bono, H., T. Kasukawa et al: READ – RIKEN Expression Array Database. Nucleic Acids Research 30(1), 2002

[BR02] Brazma, A., A. Robinson, J. Vilo: Gene Expression Data Mining and Analysis. In Jordan, B. (Ed.): DNA Microarrays - Gene Expression Applications, Springer, 2002

[BV00] Brazma, A., J. Vilo: Gene Expression Data Analysis. FEBS Letters 480, 2000

[CD02] Crom, S.L., F. Devaux et al: yMGV – Helping Biologists with Yeast Microarray Data Mining. Nucleic Acids Research 30(1), 2002

[CP01] Cornell, M., N. Paton et al: GIMS – A Data Warehouse for Storage and Analysis of Genome Sequence and Functional Data. Proc. 2nd IEEE International Symposium on Bioinformatics and Bioengineering, 2001

[CW02] Cheung, K.H., K. White et al: YMD: A Microarray Database for Large-scale Gene Expression Analysis. Proc. American Medical Informatics Association 2002 Annual Symposium

[Do02] Dopazo, J.: Microarray data processing and analysis. In Lin, S.M, Johnson, K.F. (Ed.): Microarray Data Analysis II, 43-63, Kluwer Academic, 2002

[DY02] Dudoit, S., Y.H. Yang et al: Statistical Methods for Identifying Differentially Expressed Genes in Replicated cNDA Microarray Experiments. Statistika Sinica 12(1), 2002

[ED02] Edgar, R., M. Domrachev, A.E. Lash: Gene Expression Omnibus - NCBI Gene Expression and Hybridization Array Repository. Nuleic Acids Research 30(1), 2002

[ER98] Ermolaeva, O., M. Rastogi et al: Data Management and Analysis for Gene Expression Arrays. Nature Genetics 20, 1998

[FH01] Fellenberg, K., N.C. Hauser et al: Correspondence Analysis Applied to Microarray Data. Proc. National Academy of Science 98(19), 2001

[FH02] Fellenberg, K., N.C. Hauser et al: Microarray Data Warehouse Allowing for Inclusion of Experiment Annotations in Statistical Analysis. Bioinformatics 18(3), 2002

[GL01] Gardiner-Garden, M., T.G. Littlejohn: A Comparison of Microarray Databases. Briefings in Bioinformatics 2(2), 2001

[GO00] The Gene Ontology Consortium: Gene Ontology - Tool for the Unification of Biology. Nature Genetics 25, 2000

[HB01] Hill, A.A., E.L. Brown et al: Evaluation of Normalization Procedures for Oligonucleutide Array Data Based on Spiked cRNA Controls. Genome Biology 2(12), 2001

[HB02] Hubbard, T., D. Barker et al: The Ensembl Genome Database Project. Nucleic Acids Research 30(1), 2002

[Hu01] Huyn, N.: Scientific OLAP for the Biotech Domain. Proc. 27th Intl. VLDB Conference, 2001

[HW02] Haverty, P.M, Z. Weng et al: HugeIndex – A Database with Visualization Tools for High-density Oligonucleotide Array Data from Normal Human Tissues. Nucleic Acids Research 30(1), 2002

[KG00] Kanehisa, M., S. Goto et al: The KEGG Databases at GenomeNet. Nucleic Acids Research 30(1), 2000

[KM00] Kerr, M.K, M. Martin, G.A. Churchill: Analysis of Variance for Gene Expression Microarray

Data. Journal of Computational Biology 7(6), 2000

[Kn02] Knudsen, S.: A Biologist Guide to Analysis of DNA Microarray Data. Wiley-Interscience. New York, 2002.

[KS02] Kent, J., C.W. Sugnet et al: The Human Genome Browser at UCSC. Genome Research 12, 2002

[LD96] Lockhart, D.J., H. Dong et al: Expression Monitoring by Hybridization to High-density Oligonucleotide Arrays. Nature Biotechnology 14, 1996

[LL02] Liu, G., A.E. Loraine et al: NetAffx – Affymetrix Probeset Annotations. Proc. ACM SAC 2002

[LP92] Liang P., A.B. Pardee: Differential Display of Eukaryotic Messenger RNA by means of the Polymerase Chain Reaction. Science 257, 1992.

[MC01] Mangalam, H., G. Chen et al: GeneX: An Open Source Gene Expression Database and Integrated Tool Set. IBM System Journal 40(2), 2001

[Na99] Nature Genetics Supplement 21(1), 1999

[NB98] Nadkarni, P.M., C. Brandt: Data Extraction and Adhoc Query of an Entity-Attribute-Value Database. Journal of the American Medical Informatics Association 5, 1998

[Pa02] Pan, W.: A Comparative Review of Statistical Methods for Discovering Differentially Expressed Genes in Replicated Microarray Experiments. Bioinformatics 12, 2002

[PK00] Paton, N.W., S.A. Khan et al: Conceptual Modelling of Genomic Information. Bioinformatics 16(6), 2000

[PM01] Pruitt, K., D.R. Maglott: RefSeq and LocusLink – NCBI Gene-centered Resources. Nucleic Acids Research 29(1), 2001

[QC01] Quackenbush, J., J. Cho et al: The TIGR Gene Indices: Analysis of Gene Transcript Sequences in Highly Sampled Eukaryotic Species. Nucleic Acids Research 29(1), 2001

[SB00] Schuchhardt, J., D. Beule et al: Normalization Strategies for cDNA Microarrays. Nucleic Acids Research, 28(10), 2000

[SC02] Su, A.I., M.P. Cooke et al: Large-scale Analysis of the Human and Mouse Transcriptomes. Proc. National Academy of Science 99(7), 2002

[SH01] Sherlock, G., T. Hernandez-Boussard et al.: The Stanford Microarray Database. Nucleic Acids Research 29(1), 2001

[SP01] Stoeckert, C., A. Pizarro, et al: A Relational Schema for Both Array-based and Sage Gene Expression Experiments. Bioinformatics 17(4), 2001

[SS95] Shena, M., D. Shalon et al: Quantitative Monitoring of Gene Expression Patterns with a complementary DNA Microarray. Science 270, 1995

[SW95] Somogyi, R., X. Wen et al: Developmental Kinetics of GAD Family mRNAs Parallel Neurogenesis in the Rat Spinal Cord. Journal of Neuroscience 15(4), 1995

[VE98] Vasmatzis, G., M. Essand et al: Discovery of Three Genes Specifically Expressed in Human Prostate by Expressed Sequence Tag Database Analysis. Proc. National Academy of Science 95, 1998

[VZ95] Velculescu, V.E., L. Zhang et al: Serial Analysis Of Gene Expression. Science 270, 1995

[WC02] Wheeler, D.L., D.M. Church et al: Database Resources of the National Center for Biotechnology Information: 2002 Update. Nucleic Acids Research 30, 2002

Dissertations-Preise

Quality of Service and Optimization in Data Integration Systems

Reinhard Braumandl*
University of Passau

Abstract: Due to the ever increasing impacts of globalization, people will/have to work on data which is distributed all around the world [LKK+97]. Query processing on the corresponding data sources is a key data processing task in most distributed applications and rather difficult to implement for interactive usage in an Internet scale environment. ObjectGlobe is a data integration system which enables interactive query processing in such an environment. The basic architecture of the ObjectGlobe system and its integrated quality of service (QoS) management component were developed in this dissertation and briefly summarized in this paper. The dissertation [Bra02] itself can be downloaded at
`http://elib.ub.uni-passau.de/opus/volltexte/2002/27`.

Advisors: Prof. Alfons Kemper, Ph.D. (University of Passau)
Prof. Dr. Donald Kossmann (Technical University of Munich)

1 Introduction

The proceeding globalization we encounter today results in a demand for global data processing. The Internet as the largest global computer network is the most prominent enabler for global data processing. Query processing is one task in this field and it plays an important role for many application domains, e.g., applications which depend on some kind of decision making. In this respect, the Internet provides access to a large number of interesting data sources like hotel listings, flight schedules, stock rates, earth observation images or genome databases. Data integration or mediator systems were developed to access these distributed and heterogeneous data sources on the Internet. First, these systems were closed systems with a predetermined, fixed set of data sources and a centralized architecture with respect to the execution of mediator related tasks. More recently, flexible, distributed data integration systems were conceived, e.g., Amos II [JKR99] or our ObjectGlobe system [BKK+01]. In the following paragraphs, we describe the ObjectGlobe architecture which represents an open and distributed data integration system.

Within an ObjectGlobe federation three kinds of providers that can participate in a corresponding information economy are distinguished: *data providers* which supply data, *func-*

*current affiliation: Allianz Versicherungs AG

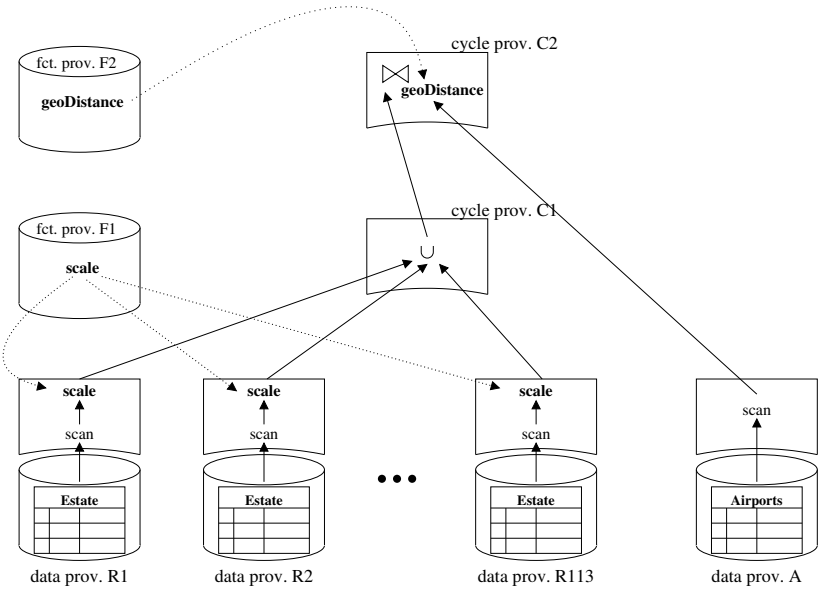


Figure 1: An Example Query Execution in an ObjectGlobe Federation.

tion providers which offer operators and functions to process data, and *cycle providers* which are contracted to execute operators and functions. In ObjectGlobe, the services of these kinds of providers can be combined in an almost orthogonal way; exceptions just arise for security, privacy and capacity reasons. Therefore, ObjectGlobe enables an open and distributed query processing services market.

For example, suppose that an ObjectGlobe user is searching for a villa in the Mediterranean area with requirements regarding the maximum distance to the next airport and the amount of building area. The notional SQL-query for this application computes a join between real estate information and data about airports. The join predicate uses a user-defined, external function which computes the length of the bee-line between two locations. The query also uses an external operator (named **scale**) for scaling the images of the estate offers into a handy size. The meaning of the attributes of our real estate relation should be obvious.

```
select e.Price, e.Location.City,
       scale(e.Image,0.3), a.Name
from Estate e, Airports a
where e.building-area > 200 and
  geoDistance(e.Location,a.Location) < 10
  and e.Location.region = 'Mediterranean';
```

In an ObjectGlobe federation, external data sets are incorporated by so-called wrappers and internally represented in a nested relational format. Hence, the data of each European realtor could be incorporated as a partition of such a relation, here named *Estate*. An example query execution in an ObjectGlobe federation is shown in Figure 1. The data

for airports is contributed by a data provider **A** and for real estate by the data providers **R1, . . . , R113**, respectively. The providers for real estate information represent combined data/cycle providers, executing the external function **scale** which is provided by the function provider **F1**. Additionally, two pure cycle providers execute the **union** operation on the real estate partitions and the **join** operation between real estate and airports information. Pure cycle providers play an important role when data providers cannot or are not willing to execute any query operators except the scan operators or when operations can be placed at cycle providers in a way which reduces communication costs.

Obviously, the effects of query executions in such an environment can hardly be overseen by a user. Therefore, a quality of service management component of ObjectGlobe supports user defined quality constraints [BKK03]. [Wei99] gives a good motivation for the need to integrate the handling of service quality guarantees in information systems. One aspect of QoS in a data integration system is certainly data quality. This topic has already been tackled in the literature, for example, in [NLF99].

2 The Quality of Service Model

As we have seen in the introduction, in an information economy a user should be able to constrain the relationship between the qualities of the result and the execution itself and the costs for the services of providers. For example, users would then be able to express that they are willing to pay more for the execution of a query, if the query finishes earlier.

This means, that analogously to cost models in traditional database systems, we need a model for our quality constraints in order to describe and assess the quality of queries, query evaluation plans and query executions.

Therefore, we need a set of QoS parameters which can be used for declaratively specifying QoS constraints for a query execution. In the query processing context QoS parameters can belong to three different dimensions: the result quality of the query, the duration/timeliness of the query execution and its monetary cost.

Parameters for the Query Result Quality:

- the oldest time stamp of the last update for a partition p of of a used relation S or its maximum staleness factor.
- the share of the used partitions in respect to the whole data of a relation S. This parameter is also called completeness.
- a lower bound for the result cardinality. This parameter can be used to express that the user expects a minimum number of result tuples.
- an upper bound for the result cardinality. Such a parameter corresponds to a *stop after* clause, whose support in query optimization and execution has already been studied in the literature [CK98].

Parameters for the Query Execution Time: The execution of the query is characterized by the time spent in different phases of the execution of a plan:

- the time spent in the `open`-phase of a plan. In an iterator based [Gra93] query engine like ours, this is roughly the time from the start of the query execution until the first tuple can be delivered.
- the time for producing all the result tuples of the plan starting at the point, when the open-phase has finished and ending, when the last tuple has been produced by the query. This phase is called the `next`-phase.

Parameters for the Query Evaluation Cost: Since providers can charge for their services in our information economy, a user should be able to specify an upper bound for the respective consumption by a query. Therefore, the quality parameters regarding the cost of a query take into account:

- the cost for services of function providers, i.e., the cost for leasing a function for the duration of the query.
- the cost for services of data providers, i.e., the cost for reading the data at the respective data providers.
- the cost for services of cycle providers, i.e., the cost for executing parts of the query at foreign cycle providers.

3 The Integration of QoS Management in Query Processing

Naturally, the ability to guarantee QoS constraints depends on the service quality guarantees one can get from the underlying shared resources. Among the common scheduling disciplines *best effort*, *priority-based scheduling*, and *reservation*, only the latter allows to construct a QoS management which can absolutely guarantee the QoS constraints for an accepted query. But reservation normally entails over-booking and inefficient resource utilization and is therefore rarely used for scheduling computer or network devices. Due to this fact, there remain two obvious goals for QoS management in our context:

- The percentage of queries, whose quality constraints could be fulfilled, should be maximized. This percentage is calculated based on the overall number of queries which are issued and not only on the number of those for which a constraint compliant query plan could be determined.
- The execution of queries which cannot fulfill their QoS constraints, should be stopped as early as possible. In this way the query does not waste the time an money of the user anymore. Of course, if the missed quality constraint is a soft one, the query should not be stopped but executed in a best-effort manner.

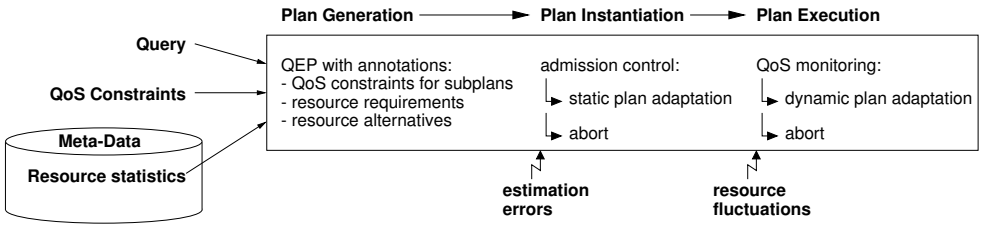


Figure 2: The Interaction of Query Processing and QoS Management

An overview of query processing with integrated QoS management is depicted in Figure 2. The starting point for query processing in our system is given by the description of the query itself, the QoS constraints for it and statistics about the resources which can be used for processing the query. In our scenario cycle providers, the partitions from data providers and the functions of function providers belong to these resources together with all the network links connecting them.

Figure 2 also shows the activities of our QoS management during the query processing phases of plan generation, instantiation and execution.

Plan Generation: The optimizer generates a query evaluation plan (QEP) which contains information about the used data, cycle and function providers and about the way their services are combined to compute a specific query. The optimizer itself is in essential an intelligent search routine which is in many cases and also in ours dynamic programming based. It assesses a huge number of different plans with different providers by a quality model which provides formulas for estimating the quality parameters based on the structure of the plan and statistics about the providers. Hereby, every considered plan is constructed piecewise in a bottom-up manner and for every sub-plan which appears in such a process its quality parameters are computed with the quality model. Only a plan which fulfills all the user constraints is considered for the later phases and its plan description will be annotated with the quality estimations and resource requirements for every sub-plan. Additionally, if the optimizer can find approximately equivalent alternatives for resources used in the query evaluation plan, these are also annotated in the plan.

Plan Instantiation: During plan instantiation, sub-plans are distributed to cycle providers, functions are loaded from function providers and connections to data providers are established. When a sub-plan of a query uses the service of a specific provider, it is checked, if the resource requirements resulting from the quality constraints for that sub-plan can be satisfied by this provider. For a cycle provider this would mean that if the optimizer underestimated the load on the respective cycle provider the newly arrived sub-plan will probably not be able to meet its constraints. Furthermore, all the other sub-plans on that cycle provider would be in danger of missing their quality constraints, if we execute the new sub-plan there. As a result of this admission control, the execution of the new sub-plan would be rejected or, if possible, the

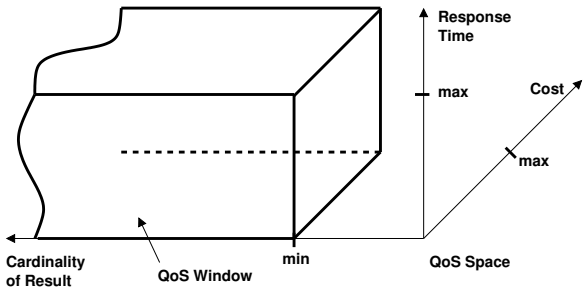


Figure 3: The QoS Space and the QoS Window.

sub-plan will be adapted.

Plan Execution: During query execution, fluctuations regarding resource availability for, e.g., CPU time or network bandwidth and again estimation errors by the optimizer might violate the constraints on the quality parameters. In order to detect these violations, a monitoring component traces the current status of these parameters for every relevant sub-plan of the query. If this component detects a potential violation of the quality constraints for a sub-plan, it first tries to adapt the sub-plan so that it will meet its constraints again, or if this is not possible, it will abort the execution of this sub-plan. The plan adaptations during the instantiation phase can be performed rather easily because the plan is not instantiated yet. Here we need adaptations which can be applied also after a sub-plan has started to execute.

By estimating the necessary quality constraints for sub-plans of a QoS compliant query plan the QoS management can monitor the development of the quality parameters on a fine granularity. This helps in detecting potential quality misses quite early. For example, if there are pipeline-breaking operators in the plan, most of the work for the query could have already been done, when the first tuples arrive at the top of the plan. In our case, the plan beneath the pipeline breaker has its own quality constraints which must be monitored and enforced by the QoS management.

4 Quality of Service Enhanced Plan Generation

In this section we mention the necessary modifications of a classic, dynamic programming based query optimizer for supporting QoS constraints during plan generation. We concentrate on the description of those parts of the optimization process which play an important role for QoS management and thus need modifications compared to their standard form.

Selecting Providers In many cases, it will not be feasible to consider all possible data providers for a given data set because the resulting amount of data processed in a widely

distributed environment would result in unacceptable running times. Thus, an additional task for QoS management is to select the most relevant data providers and find appropriate cycle providers which are able to efficiently process the data from the selected data providers. Since compact query evaluation plans will also be rather efficient in a large scale environment, we use clustering algorithms to group data and cycle providers with respect to their 'network distance' to each other. During optimization it is often sufficient to work on the resulting clusters instead of single providers. In this way, we can avoid the combinatorial explosion in finding appropriate subsets of providers.

Estimating QoS Parameters Query evaluation plans are constructed in a modular way using basic operators such as join, union, or user-defined operators. Analogously, the cost (and other properties) of plans are computed in a modular way using cost functions for the individual operators. For QoS management, an extended framework is needed in order to construct plans and estimate the properties of plans in such a modular way. In our framework, these computations are based on formulas for every operator which determine the amount of work the operator has to perform at the phases given by the iterator model. The information about the distribution of operators and parallelism properties of the plan construction are then used to compute the corresponding effects on the overall timing of a plan.

Pruning Query Evaluation Plans The query optimizer enumerates alternative plans and prunes inferior plans based on their properties (e.g., estimated cost). A QoS-enhanced optimizer requires a special pruning metric in order to take all QoS parameters of a query into account. This metric is needed because we inherently use multi-objective optimization in the QoS case. The quality dimensions span a space which we call QoS space, and the user-defined constraints determine an area in that space which we call QoS window. This is shown in Figure 3 for the (simplified) three dimensional QoS space. During optimization every enumerated plan is mapped on a point in that QoS space by estimating the value for every quality parameter which appears in the quality model. Only plans which lie within the QoS window, fulfill the constraints of the user.

5 Adaptation of a Query Execution Plan

Query evaluation plans in our system have a "natural" fragmentation which is given by the thread and machine boundaries which appear in the instantiated operator tree. At these boundaries, monitor operators trace the actual quality parameters of their input plans and forecasts these parameters for the end of execution. The corresponding optimizer estimates for the respective sub-plan (as shown in Figure 4) represent the target values for the forecasted values and a comparison of these values shows if the corresponding sub-plan is still within its quotas.

When a violation of the QoS constraints is expected during the runtime of a query, we can try to counteract this violation by adapting the query execution plan. The adapta-

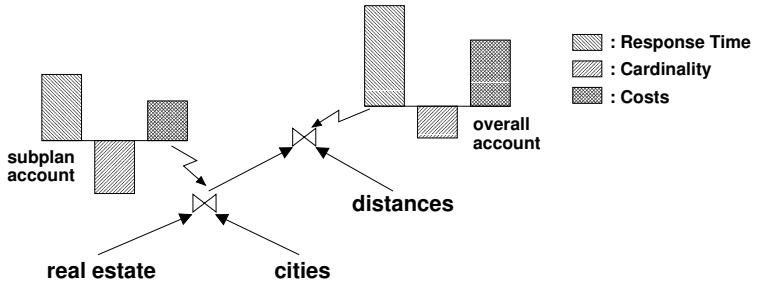


Figure 4: QoS Accounts of a Query Plan.

tions which we employ in our system to react to predicted QoS violations, operate on the resource allocation of the query evaluation plan. The corresponding resources are, for example, cycle providers, partitions from data providers and functions from function providers.

Example adaptations are:

increasePriority/decreasePriority adapt the priorities of threads which execute queries on ObjectGlobe servers. By this adaptations, we can speed up queries which need to finish sooner and we can slow down queries which currently seem to meet their time limits easily and can therefore free resources for queries with critical constraints.

useCompression activates on the fly compression for network links which seem to be slower than expected.

movePlan can be used to move a sub-plan of a query from one cycle provider to another cycle provider. Again, this adaptation can be performed on the fly in an ObjectGlobe federation, even if the plan has already begun to work.

6 Controlling Adaptations

Plan adaptation during runtime is controlled by monitor operators because they are placed at the most interesting positions for adaptations in the query evaluation plan and also gather most of the information which is needed for this task. As shown in Figure 5, a monitor operator uses a rule-based fuzzy controller which gets the forecasts of the quality parameters and other state descriptions of the respective plan fragment as input. With this information the fuzzy controller determines, if an adaptation should be applied and what adaptation this should be.

One reason for the use of a fuzzy controller in our system is that our runtime adaptations are discrete and this is quite easy to support in the inference rules which form the basis of the fuzzy controller's decisions. Furthermore, estimation errors and resource fluctuations introduce some uncertainty factors in the control process which can be modeled by

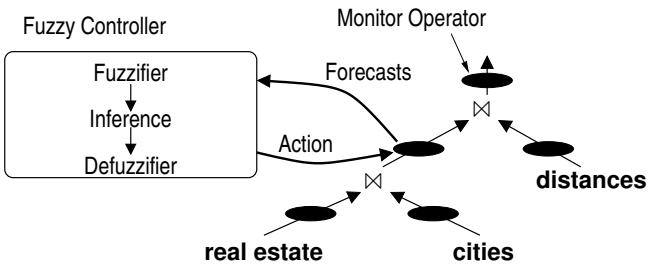


Figure 5: Feedback Loop for QoS Adaptations.

fuzzy logic [KY95] quite easily. The inference rules of a fuzzy controller also provide a highly configurable means to incorporate expert knowledge for the adaptation process in a very intuitive way. Therefore, it becomes possible to experiment with varying adaptation strategies by just changing the inference rules and perhaps the definition of the underlying fuzzy sets.

A fuzzifier transforms the numeric input values into *linguistic values* of *linguistic variables*. For each input variable of the controller there exists one such *linguistic variable*. This transformation is depicted for the forecasted cost parameter in Figure 6. The inference rules of a fuzzy controller works on linguistic variables and values. Such a rule is of the form

if X_1 is A_1 and $\ldots$ and X_n is A_n **then** Y is B

where $X_1, \ldots, X_n$ and Y are linguistic variables and $A_1, \ldots, A_n$ and B are linguistic values with accompanying fuzzy sets. A small portion of an example rule set is shown below. Here *lfc*, *lfr* and *lft* are the linguistic variables for cost-, result- and time quality parameters; *lep*, *lbp* and *lss* correspond to the execution progress, the buffer pressure and the state size.

if *lfc* is *endangered* and *lep* is *late* **then** *abort* is *preferred*

if *lft* is *endangered* and *lbp* is *high* and *lss* is *small* **then** *movePlan* is *preferred*

if *lft* is *endangered* and *lep* is *middle* and *lbp* is *high* **then** *increasePriority* is *possible*

if *lfr* is *endangered* and *lep* is *early* and *lfc* is *compliant* **then** *addSubPlan* is *preferred*

7 Acknowledgements

First of all, I have to thank my advisors Prof. Alfons Kemper and Prof. Donald Kossmann for their support. They gave me the opportunity to participate in an ambitious and visionary project. I have learned a lot from their insight and experience in doing research work. Their advices provided invaluable guidance for my work.

I also wish to express my gratitude to the whole ObjectGlobe team at the University of Passau for many helpful discussions and for the pleasant working atmosphere

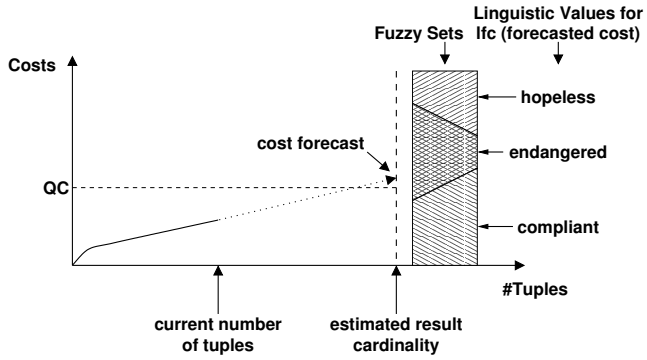


Figure 6: Mapping Quality Parameter Forecasts on Fuzzy Sets.

Literaturverzeichnis

[BKK+01] R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker. ObjectGlobe: Ubiquitous query processing on the Internet. *The VLDB Journal: Special Issue on E-Services*, 10(3):48–71, August 2001.

[BKK03] R. Braumandl, A. Kemper, and D. Kossmann. Quality of service in an information economy. 2003. Submitted for publication.

[Bra02] R. Braumandl. *Quality of Service and Optimization in Data Integration Systems*. PhD thesis, Universität Passau, Fakultät für Mathematik und Informatik, D-94030 Passau, 2002. Universität Passau.

[CK98] M. Carey and D. Kossmann. Reducing the braking distance of an SQL query engine. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 158–169, New York, USA, August 1998.

[Gra93] G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[JKR99] V. Josifovski, T. Katchaounov, and T. Risch. Optimizing queries in distributed and composable mediators. In *Proc. of the IFCIS International Conference on Cooperative Information Systems*, pages 291 – 302, Edinburgh, Scotland, 1999.

[KY95] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic*. Prentice Hall, 1995.

[LKK+97] P. Lockemann, U. Kölsch, A. Koschel, R. Kramer, R. Nikolai, M. Wallrath, and H.-D. Walter. The network as a global database: Challenges of interoperability, proactivity, interactiveness, legacy. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 567–574, Athens, Greece, August 1997.

[NLF99] Felix Naumann, Ulf Leser, and Johann Christoph Freytag. Quality-driven integration of heterogenous information systems. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 447–458, Edinburgh, GB, September 1999.

[Wei99] G. Weikum. Towards guaranteed quality and dependability of information services. In *Proc. GI Conf. on Database Systems for Office, Engineering, and Scientific Applications (BTW)*, Informatik aktuell, New York, Berlin, etc., 1999. Springer-Verlag.

Data Warehouse Schema Design*

Jens Lechtenbörger
Dept. of Information Systems
University of Münster
Leonardo-Campus 3
D-48149 Münster, Germany
lechten@wi.uni-muenster.de

1 Introduction

A data warehouse is an integrated database primarily used in organizational decision making. Although the deployment of data warehouses is current practice in modern information technology landscapes, the methodical schema design for such databases has only been studied cursorily. In this paper we pursue schema design for data warehouses in the spirit of classical database design, organized as a sequence of *requirement analysis and specification* to collect user requirements, *conceptual design* to model the data warehouse as a multidimensional database independently from implementation issues, *logical design* to transform the conceptual data warehouse schema into the target data model, and *physical design* to maximize performance with respect to a target database system.

In short, from a conceptual point of view a data warehouse is a multidimensional database, and *fact schemata*, such as the one shown in Figure 1, represent such databases conceptually. In Figure 1, we have a fact schema `Account` from the banking domain, where *measures* `Balance`, `BalanceClass`, and `NoOfTransactions` are shown in a two-dimensional context of *dimensions* `Time` and `Account`. Each dimension is specified by means of a lattice of *dimension levels*, whose bottom element is called *terminal* dimension level (here: `Day`, `AccID`). Importantly, domains of *optional* dimension levels may contain inapplicable null values, and *context dependencies* specify *contexts of validity* for optional dimension levels (e.g., `Age` is applicable to private customers, whereas `LegalForm` is applicable to capital companies, and annotations in the schema indicate these facts).

Given a fact schema F, we call the attributes occurring in F the *universe of* F, denoted by U_F, and we note that there is a set of functional dependencies over U_F, called the *functional dependencies implied by* F, denoted by FD_F (e.g., the set of terminal dimension levels functionally determines the set of measures, and each edge in a dimension lattice indicates a functional dependency).

By contrast, from a technical point of view a data warehouse for one or more (operational)

*This paper is an extended abstract of [Lec01].

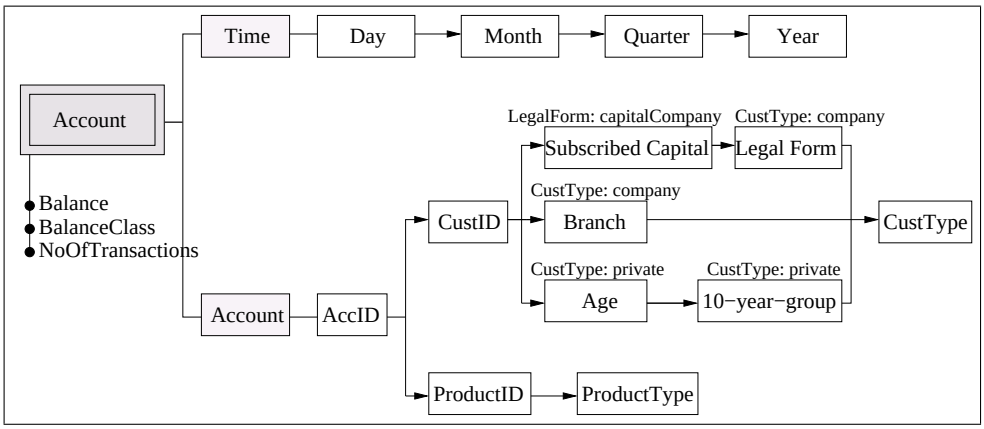


Figure 1: Sample conceptual schema.

data sources is a separate database consisting of a set of materialized views that integrate data from the sources. For our purposes, we view a data warehouse as a set of UPSJR views, i.e., relational views defined by union, projection, selection, join, and renaming.

We restrict our attention to the design process up to and including logical design, the relational data model as the target model to be used during logical design, and we emphasize that we do not address design issues that are related to, e.g., cleansing, maintenance, or meta-data management. Our aim is to define a data warehouse design process based on the *extension* of independently developed data warehouse design approaches and their *integration* into the traditional database design process. For this purpose, we strive (1) to identify and formalize desirable warehouse schema properties as design goals and (2) to set up conceptual and logical design phases in such a way that the identified design goals are guaranteed to be fulfilled.

The remainder of this paper is organized as follows. In Section 2, we present the design goals of multidimensional normal forms and independence, which are addressed by a design process outlined in Section 3. In Sections 4, 5, and 6, we summarize requirement analysis, conceptual design, and logical design as phases of this design process, and in Section 7 we sketch a method to enforce update independence in the course of logical design. We conclude the paper in Section 8.

2 Design Goals

While normal forms have a long tradition in the area of relational databases as guidelines for good schema design, research on quality factors for data warehouses or multidimensional databases has started only recently.

2.1 Summarizability and Multidimensional Normal Forms

The work presented by Lenz and Shoshani [LS97], which provides necessary conditions for summarizability, can be regarded as a first step to define quality factors for conceptual data warehouse schemata. The notion of *summarizability* refers to the possibility to compute aggregate values with a lower level of detail from aggregate values with a higher level of detail. Roughly speaking, summarizability is given if an analyst is guaranteed to obtain consistent results when performing sequences of roll-up operations along a path in a dimension lattice. Lenz and Shoshani [LS97] argue that summarizability is of most importance for queries concerning multidimensional data. Hence, any multidimensional schema should be set up in such a way that summarizability is obtained to the highest possible degree, and violations of summarizability should be expressed in the schema. Following and extending the summarizability results presented in [LS97], Lehner et al. [LAW98] define the *generalized multidimensional normal form* (GMNF), which ensures summarizability in the presence of optional dimension levels in a context-sensitive manner and supports an efficient physical database design.

We extend the framework of [LAW98] by showing that there is a natural correspondence between optional dimension levels and attributes occurring in sub-classes of generalization hierarchies in the sense of Smith and Smith [SS77], and we define three increasingly restrictive multidimensional normal forms to address this correspondence (see also [LV02b]).

We briefly summarize the impact of those three multidimensional normal forms. Let F be a fact schema with universe U_F, and let FD_U be a set of functional dependencies over U_F that can be observed in the application domain. Associated with F there is a second set of functional dependencies over U_F, namely the functional dependencies implied by F, FD_F. We argue that F can only be regarded as reasonable schema for the application domain if certain relationships among FD_U and FD_F hold, and these relationships are made formally precise in terms of first multidimensional normal form (1MNF). Roughly, 1MNF demands that (1) the functional dependencies FD_F do occur in the application domain, (2) the potential for roll-up and drill-down operations available in the application domain is preserved in the fact schema, and (3) the fact schema embodies the multidimensional contexts of measures.

While the first of these requirements is a natural *semantic faithfulness* condition that should be satisfied in any database, the following one is specific to the context of multidimensional data, where it formalizes a *completeness* aspect with respect to coverage of the underlying application domain. Finally, it can be shown that the last of these requirements *avoids* certain kinds of *redundancies* that arise due to transitive functional dependencies; hence, this last requirement sets up a loose connection between traditional normal forms and multidimensional ones.

Next, 2MNF strengthens 1MNF by taking care of optional dimension levels in order to guarantee *summarizability* in a context-sensitive manner, just as GMNF does. Thus, if a schema is in 2MNF then an analyst (or an analysis tool for that matter) can identify optional dimension levels based on schema information, which has two consequences.

(1) If an analyst considers measures in conjunction with optional dimension levels, then she is aware of the fact that she is only looking at partial information. (2) Contradictory queries (e.g., group company customers according to their jobs) can be avoided.

Finally, 3MNF places further restrictions on schemata in 2MNF, and these restrictions are sufficient to *construct* a class hierarchy concerning dimension levels, which is implicitly contained in a fact schema, in terms of relation schemata that *avoid null values*. This normal form will be our formal guideline to measure the quality of *conceptual* data warehouse schemata. In the next subsection, we turn towards properties of *logical* data warehouse schemata.

2.2 Independence Properties

From a technical point of view we regard a data warehouse as a relational database that stores a set of materialized relational views. These materialized views have to be maintained in order to keep track of updates in the operational databases, and self-maintainability has been identified as desirable property that can be exploited during view maintenance to avoid maintenance queries and update anomalies.

Roughly, a set of materialized views is *self-maintainable* if it is possible to determine the new view instance after every database update only based on the old view instance and the update information itself [GJM96]. Since a logical data warehouse schema is defined in terms of a set of materialized views, the property of self-maintainability can immediately be perceived as property of logical data warehouse schemata. In the data warehousing context this property appears particularly attractive for the following reasons: A data warehouse typically integrates information from a variety of heterogenous information sources. In such a scenario, it might already be difficult to extract deltas representing source changes, but querying the sources is almost impossible [GJM96]. Indeed, information sources may not provide query interfaces or may not permit ad-hoc queries. Even if querying of sources is possible then the associated maintenance algorithms can incur processing delays with undesirable loads on operational systems, and such queries can cause maintenance anomalies. Clearly, these problems do not occur if the data warehouse is self-maintainable.

So far, we have argued that a data warehouse should be self-maintainable. According to the terminology introduced by Laurent et al. [LLSV01], self-maintainability is also called update independence, because a self-maintainable data warehouse is independent from the information sources as far as the consistent integration of updates is concerned. Moreover, the concept of update independence can be extended to queries as well. Intuitively, a data warehouse is query-independent, if every query to the sources can be answered using the data warehouse views *only*. The motivation for enabling data warehouses to answer queries that could also be posed directly to the sources is similar to the motivation for update independence, i.e., direct querying of sources may be impossible or too expensive. Therefore, a data warehouse should not only be update-independent but also query-independent with respect to the queries that are important to the warehouse users.

3 Data Warehouse Design

Traditional database design proceeds in a sequence of conceptual, logical, and physical design steps, where each step results in a corresponding database schema. Importantly, this separation of design phases allows to reason about different aspects of a database system at varying levels of abstraction. There is a growing consensus that this separation of design phases known from traditional database design is also advantageous in the context of data warehouse design. Nevertheless, existing data warehouse design processes lack clearly defined design goals. In particular, previous design processes are unaware of desirable schema properties such as multidimensional normal forms and independence. Indeed, we argue that conceptual data warehouse schemata should satisfy 3MNF, whereas logical data warehouse schemata should be at least update-independent.

In the following sections we outline a data warehouse design process that comprises the phases of traditional database design and addresses the design goals of normal forms and independence. We assume that data warehouse design starts with the conceptual design, which is divided into requirement analysis and design of the conceptual schema; afterwards, logical and physical design are carried out in separate phases. As even a cursory treatment of the physical design phase is beyond the scope of this work, we refer the reader to [BG01], where an initial set of pointers to relevant research on this subject can be found.

4 Requirement Analysis and Specification

In the first phase of data warehouse design the data requirements to be met by the forthcoming data warehouse have to be analyzed and specified. As opposed to the requirement analysis performed during traditional database design, data warehouse design aims at the integration of a number of pre-existing operational data sources. Hence, the schemata describing these sources form a major input to the requirement analysis.

In the course of the requirement analysis, data warehouse designers, warehouse end users, and business domain experts select relevant data warehouse attributes and define initial OLAP queries based on the information found in operational database schemata. We neglect a detailed description of *how* the requirement analysis can be accomplished; instead, we focus on *what* the requirement specification should deliver in order to support the schema design process. To this end, we propose to structure the requirement specification in terms of a set of initial multidimensional or OLAP queries and a derivation and usage table, which describes the identified relevant data warehouse attributes.

The multidimensional queries can be seen as yardstick for the functionality of the data warehouse under design, while the derivation and usage table contains an informal description for each identified warehouse attribute, specifies how its values are derived from the operational databases, and indicates whether the attribute may be used as measure or dimensional attribute. Afterwards, each identified warehouse attribute has to be classified either as measure or as dimension level or as property attribute. Then, to prepare the design goal of multidimensional normal forms, optional dimension levels must be distinguished

from mandatory ones, and a context of validity has to be identified for each optional level. In this respect we propose to determine a *basis* for the functional dependencies among warehouse attributes, and we show how such a basis helps to identify (a) measures, (b) certain problems related to dimension levels, and (c) contexts of validity.

5 Conceptual Design

The conceptual design phase performs a transformation of the semi-formal requirement specification into a conceptual schema using a novel, formalized multidimensional data model. The formalization results in a multidimensional diagram such as the one shown in Figure 1, which comprises fact schemata with their related measures and dimension lattices in an intuitive graphical notation. We propose an algorithmic design process to derive fact schemata starting from the requirement specification and functional dependencies of the operational schemata, and we prove that the resulting schemata satisfy 3MNF.

As shown in Figure 2, we structure the process of conceptual data warehouse design into three sequential phases: design of initial fact schemata, dimensional lattice design, and definition of summarizability constraints (see also [HLV00]).

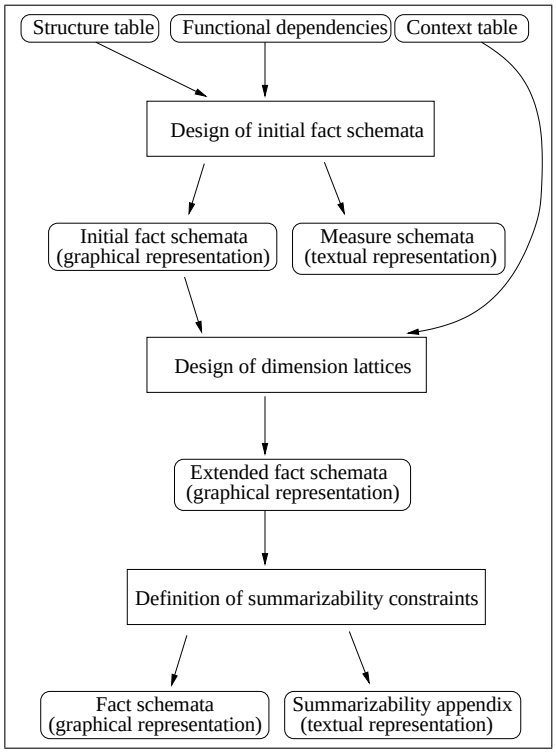


Figure 2: Conceptual schema design process.

The definition of initial fact schemata starts from the requirement specification, and the goal is to generate initial fact schemata that contain only terminal dimension levels and measures, thereby specifying the multidimensional context of measures. We just remark that this phase rests upon an analysis of functional dependencies among measures and candidate dimension levels. In the next phase, we gradually develop the dimension lattice for each terminal dimension level. To this end, we augment each initial schema with additional dimension levels by "chasing" functional dependencies starting from the terminal dimension level until no further changes arise. The final phase is about the definition of summarizability constraints. Roughly, all meaningful combinations of measures, dimension levels, and aggregate functions are enumerated.

Let F be an output schema according to the above procedure. Then we have:

Theorem 1. *F is in third multidimensional normal form.*

6 Logical Design

The conceptual schema as derived according to the previous section is now going to be transformed into a logical data warehouse schema. For this purpose, we present a transformation process that starts from a set of fact schemata in 3MNF and produces a set of view definitions which constitute an update-independent data warehouse.

The overall transformation process is sketched in Figure 3. In a first step, a relational database schema $\mathbf{D_W}$ is generated, which expresses the information modeled by the conceptual fact schemata on a logical level in terms of relation schemata and foreign key constraints. Taking advantage of input fact schemata in 3MNF, null values are avoided in relation schemata representing dimension levels. Afterwards, the resulting database schema is linked to the operational databases. To this end, a set of UPSJR views W over the operational databases is derived such that the materialization of W implements the database schema $\mathbf{D_W}$. Finally, the views W thus obtained are rendered update-independent by adding a suitable amount of auxiliary information to W, which ends the logical schema design process. Roughly, we use a so-called view complement [BS81] as key tool to derive a suitable amount of auxiliary information that renders the data warehouse update-independent. For this purpose, expressions to compute complements for UPSJR views are given. Importantly, these expressions are applicable to a larger class of views than those offered by earlier approaches and lead at the same time to uniformly smaller complements; furthermore, the complexity of constructing these expressions is shown to be polynomial in the size of schema information, which is in striking contrast to previous approaches, which are NP-complete (see also [LV02a]).

Besides, we propose a complement-based method to guarantee independence of a set of views with respect to an arbitrary set of operations, which can be applied to enforce independence properties of pre-existing warehouses or to support warehouse evolution (see also [LLSV01]).

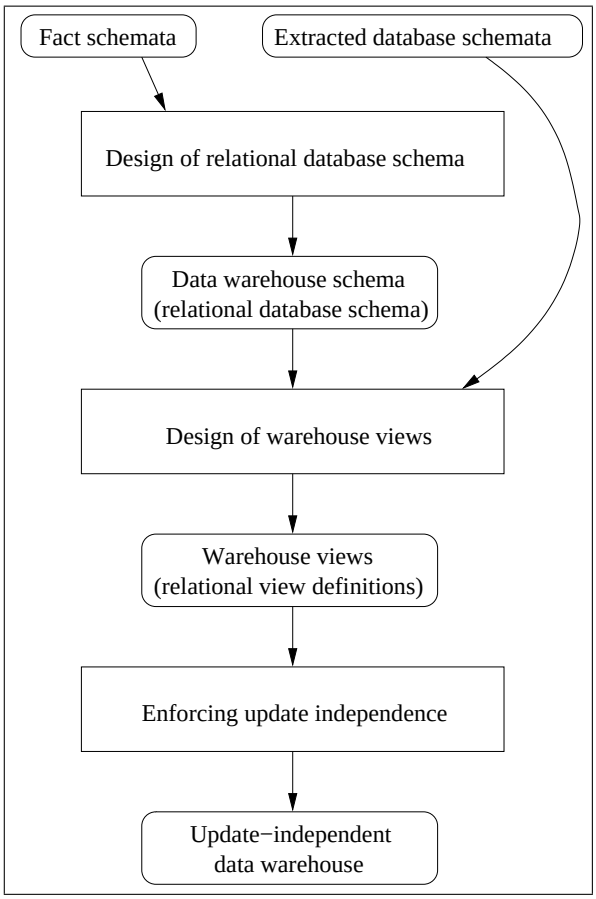


Figure 3: Logical schema design process.

7 Update Independence

We are now going to demonstrate how update independence of the data warehouse under design can be enforced in the course of the logical design phase based on the notion of view complement.

While it is well-known that single views involving projections or joins are almost never update-independent [GJM96], we now state the only known necessary condition concerning update independence:

Proposition 2. *Let D be a set of relation schemata without foreign keys, and let V be a set of views over D.*

1. *Let $V_0 =_{df} \pi_X(R) \in V$ be a view over D. If V is update-independent then it is possible to derive a duplicate counter for all tuples in instances of projection V_0.*

2. *Let $V_0 =_{df} R_1 \bowtie R_2 \in V$ be a view over D. If the domains of all key attributes of R_1 and R_2 are infinite then the following holds: If V is update-independent then the information of R_1 and R_2 is completely contained in V.*

Based on Proposition 2 we pursue the following approach to achieve the design goal of update independence by means of (a) modifying view definitions and (b) augmenting the warehouse with auxiliary views. Let D denote a set of (exported) relation schemata, and let V^0 denote a set of UPSJR views over D defining the warehouse schema.

In view of statement (1) of Proposition 2, we check for each view in V^0 whether it preserves the keys of all accessed operational relation schemata or not. If a view preserves all keys (possibly in renamed form) then it remains unchanged. Otherwise, we augment the projection with a duplicate counter. Let V^1 denote the set of views obtained from V^0 using this transformation.

To address statement (2) of Proposition 2, we add complementary views to the warehouse to ensure that relation schemata involved in joins can be computed from warehouse views. To this end, we perform the following transformation on each view $U_j \in V^1$. If U_j involves the union operation, i.e., if U_j is of the form $\bigcup_{i=1}^{u_j} V_{j_i}$, where each V_{j_i} is a PSJR view, then we change the definition of U_j into a "source-preserving union". Let V^2 denote the set of source-preserving UPSJR views obtained from V^1 using the previous transformation. We now derive a set A_{ui} of auxiliary views to render V^2 update-independent. For this purpose, we compute a complement C of V^2 with respect to D. For each relation schema R that is involved in a join in some view in V^2 we add the complementary views for R to A_{ui}.

Theorem 3. *Using the above notation, let $W = V^2 \cup A_{ui}$ be a set of views over D. Then W is update-independent.*

8 Concluding Remarks

This work advances data warehouse schema design beyond a process based on experience and rules of thumb towards a formal framework where rigorous methods are available to achieve clearly defined design goals. However, the present approach could be refined and extended in a number of aspects. First, we have neglected physical schema design, which is a mandatory part of any database implementation to optimize the logical schema for performance reasons. Second, meta-data management is largely ignored, and a systematic study concerning the co-design of warehouse schemata and meta-data repository still needs to be performed. Next, we have not paid attention to special issues that arise when temporal data is stored or analyzed, and the design of data warehouses using temporal data models does not seem to have been studied yet. Finally, we have shown how a data warehouse can be rendered update-independent based on rewritings that add duplicate counters to warehouse views and the additional storage of complementary views to deal with updates concerning join views. Additionally, Proposition 2 suggests that this additional amount of information is even necessary, if join views do not involve projections. In general,

however, it remains an open problem to determine a minimal amount of information that is necessary to render a set of views update-independent.

References

[BS81] F. Bancilhon, N. Spyratos, "Update Semantics of Relational Views," ACM TODS 6, 1981, 557–575.

[BG01] A. Bauer, H. Günzel, eds., *Data Warehouse Systeme — Architektur, Entwicklung, Anwendung*, dpunkt.verlag, 2001.

[BLT86] J.A. Blakeley, P. Larson, F.W. Tompa, "Efficiently Updating Materialized Views," Proc. ACM SIGMOD 1986, 61–71.

[GJM96] A. Gupta, H.V. Jagadish, I.S. Mumick, "Data Integration using Self-Maintainable Views," Proc. 5th EDBT 1996, LNCS 1057, 140–144.

[HLV00] B. Hüsemann, J. Lechtenbörger, G. Vossen, "Conceptual data warehouse modeling," Proc. DMDW 2000: 6.

[LLSV01] D. Laurent, J. Lechtenbörger, N. Spyratos, G. Vossen, "Monotonic Complements for Independent Data Warehouses," VLDB Journal 10 (4), Springer-Verlag, 2001, 295–315.

[Lec01] J. Lechtenbörger, *Data Warehouse Schema Design*, Inaugural-Dissertation zur Erlangung des Doktorgrades der Naturwissenschaften im Fachbereich Mathematik und Informatik der Mathematisch-Naturwissenschaftlichen Fakultät der Westfälischen Wilhelms-Universität Münster; available as volume 79 in "Dissertationen zu Datenbanken und Informationssystemen," Akademische Verlagsgesellschaft Aka GmbH, Berlin, 2001.

[LV02a] J. Lechtenbörger, G. Vossen, "On the Computation of Relational View Complements," Proc. 21st PODS 2002, 142–149.

[LV02b] J. Lechtenbörger, G. Vossen, "Multidimensional Normal Forms for Data Warehouse Design," 2002, to appear in Information Systems, Elsevier Science.

[LAW98] W. Lehner, J. Albrecht, H. Wedekind, "Normal Forms for Multidimensional Databases," Proc. 10th SSDBM 1998, 63–72.

[LS97] H. Lenz, A. Shoshani, "Summarizability in OLAP and statistical databases," Proc. 9th SSDBM 1997, 132–143.

[SS77] J.M. Smith, D.C.P. Smith, "Database Abstractions: Aggregation and Generalization," ACM TODS 2, 1977, 105–133.

Industrie-
Programm

The IOP Approach to Enterprise Frameworks

Dr. Udo Nink, Stefan Schäfer

CronideSoft AG
Dachsgang 27
D-35428 Langgöns
(udo.nink | stefan.schaefer)@cronidesoft.com

Abstract: This paper introduces the Internet Operating Platform (IOP), an enterprise framework for large scale software development. In addition to obeying to important standards (UML, XML, Java) an enterprise framework has to fulfil three basic requirements. First of all, it has to be broad and needs an elaborate architecture complementing standards and technologies rather than purely connecting them. Therefore, IOP combines UML modeling, workflow specification, code generation, run-time configuration, and component architectures. Secondly, an enterprise framework allows most developers to concentrate on business leaving technical issues to a few specialists. Therefore, IOP abstracts from underlying technologies in the areas of front-ends (HTML, XML, Java), communication protocols (FTP, HTTP, JMS, RMI), distributed components (EJB, CORBA), and persistence (virtual memory, XML, SQL92, SQL:1999). All corresponding drivers are replaceable and can even coexist. Third, an enterprise framework has to provide micro solutions on both technical and business levels. Thus, IOP provides amongst others built-in services and components like session management on a technical level and content management on a business level.

1 Introduction

One major IT problem is the fast pace of changing technologies implying frequent changes of products, applications, market requirements, and education needs. A very promising approach to keeping pace with progressing technologies is to "think in **platforms**". In the automotive industry Volkswagen has been very successful with its realization of a platform for manufacturing and selling its car types as well as those of Audi, Seat, and Skoda (belonging to the same enterprise). As a side effect to the now unified technology and product stack solutions to other problem domains come along (uniform set of skills, uniform processes, and so on).

Naturally, software industry is making every effort to force IT platforms by definition of **standards**. In the past, single companies were able to establish de-facto standards due to the wide-spread use of their products. But today, very important as well as world-wide accepted de-jure standards emerge pushed by Internet technologies (HTTP, HTML, XHTML, XML, XSL, JAVA, Servlets, JSP, EJB, JMS [W3C, OMGa, Co01, Ro99]). Moreover, with UML [RJB99] and SQL [ISO99] we share international standards for software engineering and database languages, respectively.

In addition, the Internet forces application developers to "think in **services**". Such services (like registration services) embody high reuse potential compared to customer-specific business processes or fine-grained entities.

1.1 The Need for Enterprise Frameworks

Strategic software development has to address the above mentioned key factors. Approaches to implement such a strategy range from out-of-the-box solutions to custom development. From our observations in the areas of markets, projects, applications, and technologies (in **bold-face**, see list below) the answer lies in the middle and is named enterprise frameworks. Our approach to enterprise framework is IOP, the Internet Operating Platform [Cr02, Sc03].

Markets: Out-of-the-box solutions are adequate in near-perfect-fit cases, but fail for high customization effort. On the other hand, from-scratch-solutions lead to high overhead in choosing and implementing with the right set of products and technologies. Enterprise frameworks are suited when customization is expected to make up more than 30 per cent. The wheel gets invented only once providing a set of products and technologies that fit together and all applications on top save the effort.
Projects: Project management shall keep projects in-time and in-budget. Team members have to fit certain roles, be trained and be coached according to chosen tools and development processes. An enterprise framework already provides an overall architecture along which responsibilities, development activities, and roles are identified and aligned. One framework expert suffices to train and coach 10 to 20 team members.
Applications: Risk mitigation implies that technical problems are solved first. Consequently, development focus initially lies on technical issues and shifts to business issues with each iteration. Break-even is reached earlier with enterprise frameworks because of already proven technology stacks spanning user interfaces, workflow, communication, integration, and persistence as well as ready-to-use, thoroughly tested, partial solutions.
Technologies: The fast paced creation of standards, technologies, and product versions periodically "makes rookies out of experts again". Enterprise frameworks provide a more stable development environment buffering IT evolution to a certain degree. They provide a platform for choosing among different technologies for different subjects (like communication) in different contexts (like calling services). And technologies can be tested and compared by integration into the framework.

1.2 Roadmap

In the following we will introduce and discuss IOP – our implementation of an enterprise framework. Section 2 introduces IOP due to different views on its architecture. Afterwards, IOP objects and IOP components are detailed in section 3. Section 4 describes IOP interaction, IOP workflow, and their collaboration. Section 5 deals with modeling and code generation and how these are embedded in a phase plan for developing an example application. The relationship to other work is sketched in section 6. Section 7 concludes our work and gives an outlook to future efforts.

2 Architecture

We will start with IOP's design goals. Then, we discuss IOP's architecture from different perspectives: building blocks, layering, systems, and topology. The building block view starts with a concise list of main concepts and micro frameworks. Then, the relationships between these blocks are discussed in the layer view. Afterwards, the system view gives an overview of grouping functionality into subsystems. The topology view describes the basics for installation and configuration. Finally, we compare different architectures.

2.1 Design Goals

IOP has an object-oriented distributed component architecture including methods and tools for the software development life-cycle. And it has a strong focus on standards. Three ubiquitous standards build IOP's foundation: UML, XML, and Java/J2EE:

- **UML** (Unified Modeling Language [RJB99]) is the base of visual modeling.
- **XML** (eXtendend Markup Language [Co01]) is the base of data exchange and configuration. Furthermore, it is used as alternative persistence model.
- **Java** is the programming language of choice; important **J2EE** packages [Ro99] are integrated with IOP. But, while J2EE is merely a huge unsorted box of APIs IOP delivers the glue putting the pieces together via a sophisticated architecture.

All other supported standards and technologies are replaceable throughout the framework from front-end to database:

- **Front-end**: Coupling of arbitrary front-end technologies in multi-channel fashion via IOP Interaction. Implementation proves include Java Swing, HTML, DHTML, and WML.Markup generation is supported using JSP, Servlets, and XML/XSLT.
- **Workflow**: Process definition via WPDL (Workflow Process Definition Language [WfMC]) or via UML activity diagrams (planned). IOP provides two levels of workflows: simple workflows executed on very fast core engines and complex workflows executed on dedicated workflow components on top of core engines.
- **Communication**: Different protocols encapsulated as IOP devices like HTTP, RMI, FTP, POP3, SMTP, and JMS. Such devices can be reused via drivers for component communication and for the IOP Virtual File System (VFS).
- **Component architecture**: Support for CORBA and EJB. IOP provides a common base for developing components based on CORBA, EJB, or internal IOP concepts.
- **Persistence**: Configurable persistence managers for virtual memory (VM), ORDBMS (SQL:1999), RDBMS (SQL92), and file system (XML) using persistence mappings (object-to-relational, object-to-object-relational, [Sc03, Ru01, Am99]).

IOP does extensive code generation. From the UML model IOP generates configuration information, Java code, and SQL code for the handling of objects, object references, object collections, and more. In addition, workflows are compiled to the core workflow engine. See section 5 on “Modeling and Code Generation” for a complete example.

2.2 Building Block View

IOP is made up of several building blocks which are roughly described in the following:

IOP Objects: Objects (more precise: business objects or user-defined object types) are classes in the UML model with stereotype "IOPBO" for IOP Business Object. They represent the smallest building block of IOP. Visually designed object models including direct relationships and inheritance are supported. These models define the static structure of object graphs that hold application data and provide for local-scope functionality. Objects have location-transparent identity, are type-safe, and can be versioned and referenced. Object graphs can declaratively be copied and be transformed between different representations (virtual memory, XML, SQL92, and SQL:1999).
IOP Components: IOP Components aggregate and manage IOP Business Objects. They provide for vertical business logic combined into vertical services. Components are the granule for transparent distribution and addressing as well as for integration of third-party services and applications. Moreover, they are key to optimization of performance and scalability. Finally, components encapsulate underlying component architectures and communication technologies. More than a dozen of ready-to-use or ready-to-adapt components are given ranging from id handling to content management.
IOP Design Repository: The IOP Design Repository holds all static design information extracted from the UML models by the IOP Compilers; its corresponding component is named "IOP Design Component". This information is enriched by mapping information for Java and SQL. Currently supported partial UML models are the class model and the component model. Dynamic design information is supported via workflows (see Workflow Framework below).
IOP Run-time Repository: The run-time repository is driven by the IOP Configuration Component. It holds information corresponding to topology (hardware nodes, software nodes, module nodes), configuration of components w. r. t. choices of technologies (drivers, devices, persistence, virtual file system), and initialization data ranging from passing of user data to specifications of load balancing and fallback.
IOP Service Framework: The IOP Service Framework is a collection of so-called micro frameworks that encapsulate partial and mostly technical solutions. Micro frameworks provide reusable services via stable interfaces. Currently, more than a dozen services are implemented like for localization, logging, or the virtual file system. A very important role is played by the device/driver concept which is used consistently throughout the framework. Devices encapsulate low-level APIs for close to ten different protocols like ftp and http. Drivers like a file system driver can be implemented on top of such devices. The drivers, in turn, can then be used for configuring instantiated services like file systems for components or software nodes. Since IOP supports parallel usage of different implementations at the same time you can easily construct file systems hiding different protocols behind simple folders much like in Unix operating systems.
IOP Persistence Framework: The IOP Persistence Framework provides for persistent storage of Java objects. It combines the concepts of SUN JDO [Ru01] and well-known concepts for object-relational mappings [Am99]. Four persistence mappings are available in IOP: IOP Virtual Memory Persistence, IOP XML Persistence, IOP SQL92 Persistence, and IOP SQL:1999 Persistence [Sc03]. Needed mapping information is extracted from the UML models and stored with each object type in the IOP Design

Repository and in the IOP Object Type System. Again, components can choose persistence mappings by configuration. A very nice spin-off of this approach is, e. g., that exporting database data (from a database wrapped by a database-driven component) is reduced to copying the data to a component driven by IOP XML Persistence!

IOP Interaction Framework: The IOP Interaction Framework defines the access point to an IOP system for the outside world. It is based on a service handler architecture and is responsible for converting requests to and responses from any IOP system. Encapsulated communication protocols (HTTP, JMS, RMI) and exchange formats (HTML, WML, XML, Java objects, messages) are implemented for various servers (like application servers). Implementation choices are, again, configurable.

IOP Workflow Framework: The IOP Workflow Framework is based on the reference architecture of the Workflow Management Coalition (WfMC, [WfMC]) and the corresponding OMG recommendations [OMGb]. IOP thus supports workflow specification via WPDL. Workflows (or processes) are configurable in terms of distributed execution, auditing, and statistics. We distinguish between workflow definition and workflow runtime for performance-improved representations. Workflow participants are mapped to an organizational model thus leveraging from existing organization data in business applications. Robustness and scalability are given by transaction awareness and disconnected session management. Since workflows can span many scopes and timescales (business workflow, user interaction workflow, technical workflow) we provide for a very fast core workflow engine as basis for more specialized engines (e. g. for user interaction).

IOP Integration Framework: The IOP Integration Framework collects concepts, implementations, and workflows for third-party systems integration. Integration components act as clients of such systems. Due to resulting intersections between integration and interaction, both micro frameworks share common code: low-level drivers for communication protocols, exchange formats, and implied transformations. Moreover, integration can reuse interaction by, e. g., registering an interaction listener on a message-oriented integration hub. Finally, integration components can also act as persistence managers thus providing access to integration data via ordinary business objects.

IOP Code Generation Framework: The IOP Code Generation Framework provides a set of compilers (including scanners, parsers, analyzers, and writers) supporting code generation for modeled design information. These compilers are heavily used for application development as well as for development of large portions of the framework itself. This self-reproducing feature guarantees continuous testing of IOP. All design information is stored in the IOP Design Component and the IOP Object Type System. Supported source languages are UML and WPDL and corresponding target languages are Java, SQL92, SQL:1999, XML, and IOP Workflow Format. Each generated business object for Java obeys to a set of inner interfaces implemented by corresponding classes for its entity (storing data), behavior (operations), collections (array, map), referencing (object linking), and persistence mappings. Moreover, SQL code is generated for definitions of types and tables. Additionally, DTD/XMLSchema code is generated structuring XML representation of business objects. Finally, workflows defined via WPDL are translated into IOP Workflow Format that can be handled by the IOP Core Workflow Engine.

2.3 Layer View

Let us now put the building blocks together. Figure 1 shows the layer view on IOP's architecture. The layers are: presentation layer (on top), application layer, component layer, and persistence layer. Starting left of the component layer you will find a UML

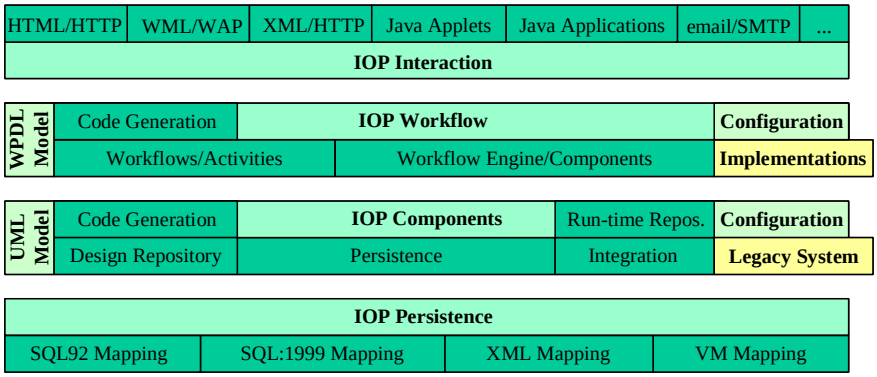


figure 1: Layer View on IOP Architecture

model containing the system design. This model is parsed and written to the IOP Design Repository. During code generation business components and internal components are generated. These can be configured to reuse existing micro frameworks like integration or persistence. The configuration (see right side) is written into the run-time repository. In case of integration a component wraps an outside system (e. g., legacy system). If a component configures a persistence manager it can choose between mappings for virtual memory persistence, xml persistence (xml files in file system), or SQL persistence (see persistence layer at the bottom).

In the application layer (2nd from top) workflows represent business processes defined in WPDL. Again, the code generator delivers internal representations. These get executed by workflow engines/components. Each such workflow consists of an activity network where activities wrap implementations. Such implementations can be external applications or IOP Executables which, in turn, coordinate calls to business components and prepare data sets for exchange with the front-end.

The presentation layer on top is served by IOP Interaction which is responsible for translating and dispatching requests and for delivering channel-specific responses. If user interaction is needed in a workflow then you define the corresponding views to be resolved by IOP Interaction as attributes to the corresponding activities. Such a view might be given by an ordinary html page uploaded into the IOP Content Component.

2.4 System View

The system view of the IOP architecture describes subsystems, that is, sets of interfaces (services) encapsulated for software nodes. The idea is to have services running in separate processes. While some subsystems are inherently needed for an IOP system others can be reused or adapted for application programming. Similar to operating systems different run levels allow for setting an IOP system to certain maintenance modes. Each (higher) run level adds functionality (e. g., run level 3 supports component startup/shutdown). Since IOP is built in Java, processes correspond to instantiated Java Virtual Machines (JVM). For the sake of performance and scalability, all needed IOP resources for JVMs are managed by the IOP Resource Manager. Here is a list of subsystems:

IOP Boot System: Starting from a single small property file various boot strap loaders are coordinated to start up or shut down the IOP kernel.
IOP Kernel System: All subsystems and any application system running on IOP are managed by the kernel system. It is responsible for switching between run levels.
IOP Configuration System: Manages IOP installations, that is, their topologies. These include hardware and software nodes, component deployment, driver choices, queries, access policies, devices, listeners, persistence, and caching.
IOP Session System: Sessions store run-time or state information during interaction with the IOP system (often user-specific information). A session system manages all sessions for a software node.
IOP Logger System: Logging is important for detecting errors and misuses. The logger system provides logging services for quickly storing log messages at certain software nodes. Log messages can be leveled (by severity) and categorized (by category and subcategory). Developers can then use output filters to quickly search for error causes when compiling or running IOP.
IOP Localization System: Localization is used in multi-language applications. It is message-based and maps message numbers to locale-specific messages. Message number ranges are also supported. Developers do only use message numbers when coding while associated message texts are defined and resolved in one place.
IOP Driver System: Many APIs (e. g., file system) and protocols (like ftp, http) can be plugged into IOP via device drivers. A driver system manages all driver instances for a software node. Each software node can thus be configured to provide certain drivers. Consequently, software nodes can be dedicated to certain APIs and protocols.
IOP Component System: Components are running in component containers. Management of component instances at run time is in the responsibility of the component system. Software nodes can be configured to provide only certain components. While drivers map APIs to protocols, components implement low-level business logic reusing drivers that abstract from technical protocols.
IOP Workflow System: Very fast core workflow engines associated, again, to software nodes execute defined workflows. Workflows can be complex (inter-department) or simple (local scope, no user interaction), Simple workflows can be run on dedicated workflow-sensitive components to improve performance or do functional enrichment.
IOP Object Location System: Objects have a logical location which is part of their OIDs. Locations map to storage managers (components in most cases). The object location system resolves locations given by OIDs (or object references).

IOP Object Type System: Only strongly typed software can be made reliable and efficient with acceptable effort. Thus, IOP forces strong typing (each object belongs to an object type). All available object types (IOP-specific and application-specific) are managed by the object type system. These are represented by meta objects for object types, components, object members, and corresponding persistence mappings.
IOP Transaction System: Critical business processes have to be transactional. Thus, the transaction system is responsible for encapsulating transaction systems like transaction monitors. IOP is based on JTS and JTA [Ro99].
IOP Statistics System: Similar to logging it is important to collect business information during system interactions. Again, storing of such information must be fast while analysis itself can be deferred and be done asynchronously. The statistics system provides services for storage and analysis.
IOP Dispatcher System: For the sake of scalability the dispatcher system delegates incoming requests to subsystems, workflow engines, and components which then become responsible for serving corresponding requests.
IOP Command System: The command system realizes a command shell to an IOP system. Command shells are well-suited for testing during development, for administration purposes, and for simple batch jobs using command scripting.

2.5 Topology View

Figure 2 shows the first part of the topology view on IOP concentrating on installation configuration as it is defined in our UML class diagrams. The configuration information is collected under an object of type IOPInstallation with stereotype IOPBO (meaning IOP Business Object). An installation pools host nodes, file systems, loggers, and components. Host nodes represent hardware (computers, processors) running software nodes (processes). Associated file systems are used for file-based input/output (e. g., a content component might import contents from a file system). Loggers are needed for storing log messages – each software node must have exactly one logger. The components in the pool can be wrapped by component instances (now see figure 3) which are associated with software nodes.

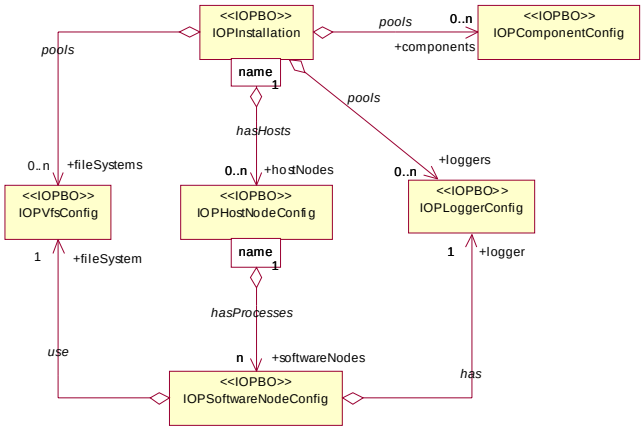


figure 2: Topology View (part 1) on IOP

The components have to be provided by component containers (like Oracle9i AS, Bea WebLogic, and Apache Tomcat). Component containers provide component servants and component drivers. Component servants only contain implemented business logic. Technology-specific code like home classes or remote classes are generically provided by IOP and, thus, need not be hand-coded by component programmers.

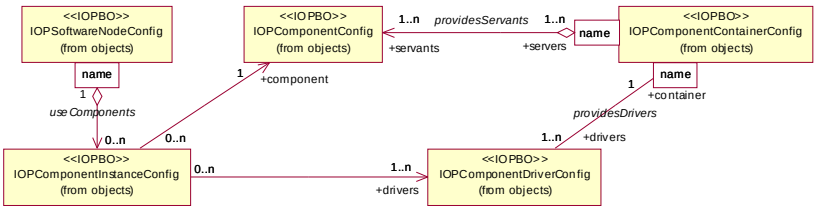


figure 3: Topology View (part 2) on IOP

Component drivers allow to communicate with components using different protocols. Redundancy and fallback (not shown here) are supported by associating redundant and fallback drivers. A software node uses components by defining component instances. Component instances link chosen component servants with chosen component drivers.

2.7 Comparison of Architectures

Figure 4 shows three different kinds of layered application architectures on the top and two example architectures at the bottom.

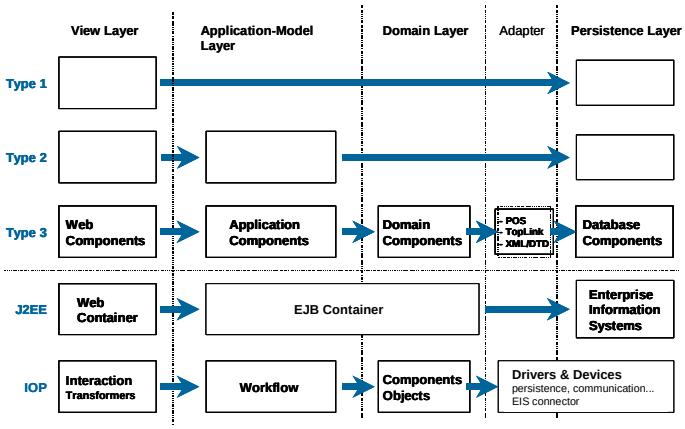


figure 4: Layered Architectures

Type 1 leads to monolithic clients mixing domain logic, application logic, and GUI logic. Type 2 introduces application components allowing for thinner clients and for reusable application logic. Type 3 enriches this model by introducing domain components collecting common application logic across different applications. Moreover, a persistence adapter layer is shown due to the technical necessity to bridge the gap from programming languages like Java to databases (impedance mismatch).

The two examples at the bottom of the figure, J2EE and IOP, are both type 3 architectures. J2EE introduces the notion of containers:

- Requests are served by web containers supporting Servlets and JSP having access to the J2EE APIs (JMS, JAAS, JTA, Java Mail, JAF, JAXP, JDBC, and connectors).
- For the application-model layer as well as for the domain layer J2EE provides EJB containers where EJBs, again, have access to the mentioned J2EE APIs.
- Access to databases (or, in general, to enterprise information systems) is typically realized using connectors, JDBC, or JMS from within EJBs of the domain layer.

IOP, first of all, allows to implement the same technical architecture as J2EE. But, in contrast, it also delivers a more process-oriented architecture as shown in the figure.

- In the left IOP Interaction serves the view layer in order to encapsulate all UI-specific communication (interaction) to applications. In addition to J2EE markup frontends IOP also supports Java Swing. In addition to http as communication protocol IOP also supports technologies like wap and smtp allowing to serve requests via mobile devices and email, respectively. A transformer approach allows to plug in channel-specific translators for requests and responses.
- For the application-model layer IOP provides IOP Workflow. Requests are mapped to workflows representing application logic and delivering responses as well as model data for driving the frontend using the MVC pattern. If by-pass is needed then requests can also be mapped directly to activities and components.
- For the domain layer IOP provides IOP Components. Implementation technologies like EJB or CORBA are hidden and can be chosen by configuration of drivers and devices. Exactly the same concept is used for connecting to databases and third-party systems.

3 Objects and Components

Section 2.2 already introduced the notions of IOP Objects and IOP Components giving a short overview over both concepts. Now, the concepts shall be detailed.

3.1 Object Types

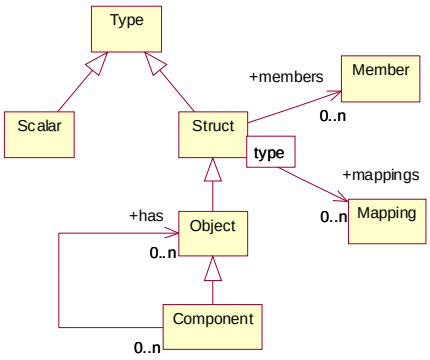


figure 5: IOP Type System Model

IOP delivers an enhanced type system having two main advantages over the Java reflection API: it is more expressive w. r. t. versions, multiplicity, containment, and persistence mapping; and it performs better (optimized member access via indices). The UML diagram in figure 5 shows our type model. The root class named “Type” is abstract and only gathers common concepts for

all kinds of types. Subclasses are “Scalar” and “Struct”. IOP provides close to 20 scalar types (like blob, boolean, byte, char, and so on). These are mapped to Java primitive types and can be used as such in a type-safe manner. When modeling, you can also choose object scalars that have been introduced to support null values. “Struct” is used for record-like structures and, thus, has members and member mappings. “Object” inherits from “Struct”. In addition, objects are identifiable, that is, have an object identification, OID in short. OIDs are globally unique identifiers and allow for location of objects and building relationships (using references) independent of their physical storage location. An OID in IOP is made up of five values: location number, type number, type version, instance number, and instance version. In addition, many operations are predefined which all IOP Objects get for free.

3.2 Object Interfaces

Modeled object types are input to the IOP code generators which deliver a set of Java classes obeying to a set of interfaces given in figure 5. On top you find the interface “Entity” collecting member setters and getters. The interface “Behavior” collects modeled and implemented operations. All other interfaces act as clients against behavior interfaces in order to use objects in different contexts. The interface “Reference” is a proxy - all object operations can be called using either the object itself or a reference to it. Dereferencing occurs transparently regardless of object locations. Furthermore, collection support is given by implemented interfaces “Collection” and “List” both physically backed up by “Array”. The interface “Map” allows for keyed access to its entries. For set-oriented operations IOP provides a universal graph interface. On top of it you get intra-object-graph navigation via object cursors and fast object graph transformation between virtual memory representation, maps, XML, and SQL.

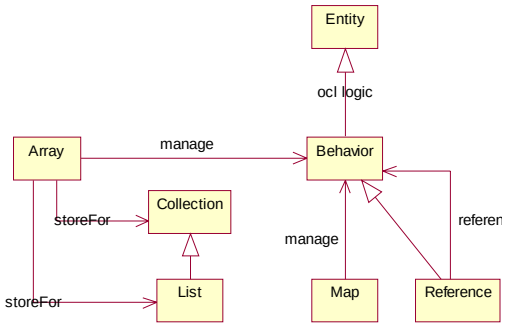


figure 5: Object Interfaces in IOP

3.3 Component Model

In the section on architecture we have discussed commonalties and differences of IOP and J2EE. Both have component models. The component model of J2EE, EJB, will be compared to IOP Components. In addition, we will give an idea on the differences to web services and what role web services can play in an IOP environment.

Since IOP allows to use EJB as technology driver for IOP Components, you can leverage from existing EJB concepts. In addition, IOP has the following main improvements:

- IOP supports EJB as well as CORBA or pure IOP Components.
- The resource management of EJB concentrates on threads, socket connections, database connections, and, of course, components. IOP also has a resource management. It knows the following resources: component containers (like tomcat for servlets or jboss for EJBs), component instances, workflow engines, and any drivers (persistence mappings, communication protocols, virtual file systems).
- IOP Components have a built-in core workflow engine that allows for execution of simple workflows. Thus, workflows can be used to chain component operations inside a component to form new operations without Java programming.
- EJB relies on RMI, RMI-IIOP, and JMS for distributed object communication. IOP additionally allows arbitrary protocols as long as drivers exist (e. g., http).
- EJBs can make use of any Java class. On the other hand, there is no global data model available across EJBs. IOP additionally supports UML class models. These can be partitioned into partial models. Each partial model can be associated to a component and, thus, become its schema. The code inside an EJB may rely only on that schema meaning that Java code as well as database queries are based on this schema. Programmers do not need to write mapping code, but rather choose persistence drivers. Of course, databases can also be directly accessed via JDBC.

Taking a look at web services we first talk about what web services are meant for, what they are, and what they are not:

- Web services are meant for communication between applications across networks and firewalls. Nevertheless, it is possible to use them for inter-component communication inside applications, too.
- Web services are useful for publication and invocation of services. They do not help you w. r. t. service complexity or assembling of services.
- Web services can be viewed as component model. In fact, they provide distributed object communication mainly by specifying component interfaces and XML messaging. But, they do not provide a programming model for implementing services or components like EJB or IOP do.

Web services are not yet part of IOP, but integration is straight forward. We view web services as yet another technology that can be configured for certain responsibilities:

- IOP can use web services via drivers for inter component communication and calls from activities to components.
- Some IOP concepts are well suited to be published as web services by generating WSDL and integrating SOAP into IOP Interaction. Candidate concepts are workflows, single activities, components, and component methods.
- IOP will not provide an UDDI implementation, since UDDI is used for global registering and finding of web services. It suffices to support access to UDDI directories.

3.4 Components

More than a dozen components are already implemented. In some cases existing IOP services are only complemented by components because of the basic need for persistent storage (left out in the list below). While some components are rather technically motivated others have been built for application development (marked "business" below).

IOP Design Component: The design component manages all elements extracted from UML modeling by model file parsers. It conforms to UML 1.3 (soon UML 1.4) with regards to class models and activity models and enriches them by needed mapping information. The so parsed models are input for the code generators for Java and SQL.
IOP ID Component: The id component is responsible for creating new OIDs with which newly inserted objects shall be stored.
IOP Workflow Definition Component: Workflow models specified via WPDL are compiled by the workflow compiler. The resulting process definitions are managed in the workflow definition component. A workflow model basically contains workflows, activities, transitions, applications, participants, and workflow relevant data.
IOP Workflow Instance Component: When instantiating a workflow its definition is first fetched from the workflow definition component. The instantiated and configured process is then stored and thus available for execution by the workflow engine.
IOP Content Run-Time Component: This component is used to optimize content management information for run-time presentation. On one hand, it provides optimized physical contents for run-time access. On the other hand, it can also provide multiple physical contents for each logical content to support multi-channeling.
IOP Content Management Component (business): The content management component supports multi-provider content sites. A site can be bulk-loaded without the need to manually specify each single content. It is basically structured into sub-sites, resources, frames, and elements. In addition to managing arbitrary files the component is also used for managing markup pages for building application front-ends.
IOP Batch Component (business): This component is not explicitly programmed. It is rather a reminder that its functionality is already given by IOP workflows. That is, workflows can be time-sensitive, and a dedicated workflow engine can be configured to control workflow execution using a clock. Consequently, periodic deliveries, nightly backups, weekend reports, and alike are built using time-sensitive workflows.
IOP Actor Component (business): All actions are performed by actors (e. g., participants perform workflows). Thus, the actor component allows for management of actors having accounts and passwords. Actors are also the basis for access restrictions managed by the access component.
IOP Access Component (business): The access component manages access control lists giving accessors access to accessibles. In most cases actors will take the role of accessors and some products will take the role of accessibles. For the sake of flexibility the access component itself does not pose any restrictions here, so that arbitrary objects may take the role of accessors or accessibles.
IOP Organization Component (business): The organization component adds roles and organizational units to actors. According to directory services or participant mappings it allows to represent complete organization structures where organizational units provide roles and manage actors taking roles.

4 Interaction and Workflow

Interaction is responsible for, basically, converting requests to and responses from an IOP system. An incoming request is translated into an IOP Message. This message is then sent to the dispatcher. The dispatcher identifies the responsible workflow that shall serve the request. The workflow (as well as implied activities, executables, and component operations) is then run until the next point of interaction. The result is, in turn, sent back as a message to the dispatcher. The dispatcher identifies the corresponding request and forwards the message to the interaction layer. The interaction layer finally translates the result message into a response leaving the system. The workflow to be executed is wrapped by a technical workflow allowing for pre- and post-processing. Thus, interaction is easily customizable by adapted technical workflows or simply by replacing the therein called executables.

Workflows are defined using WPDL. The definition spans complete workflow models including workflows, activities, transitions, applications, participants, and workflow relevant data. Unfortunately, many things have not been taken into account when WPDL was defined by WfMC – it is an open issue where to put configuration information for workflow models. For instance, you somewhere have to specify communication channels. As a result, workflow system vendors heavily use the so-called extended attributes (freely definable lists of name/value pairs) in order to backpack configuration information to the workflow definitions. Consequently, interoperability between workflow systems gets a lot harder.

During workflow execution instantiated workflow definitions are kept in the workflow instance component. An instance holds run-time information for exactly one associated run of a workflow. This run-time information is fed into the core workflow engine. The workflow system supports transactional workflows and disconnected sessions in order to make workflows reliable and to handle interruptions in case of manual activities.
Participants in a workflow model have to be mapped to persons or systems (actors in our case). Again, the WfMC leaves it to you, but at least recommends the use of organizational models. IOP therefore supports mapping of participants to actors, roles, and organizational units.
Participants might be involved as performers of several workflows and activities. Thus, work lists are supported that collect all work items (workflows and activities) a participant is responsible for. Such work lists are the base for pull or push approaches where participants or the system decide when to work on which item.

5 Modeling and Code Generation

IOP supports full project life cycles and corresponding development processes (methodologies). But, instead of defining yet another development process we rather define steps that you have to undertake during development (what is done by whom when and why). Since theory of development processes is a huge area we decided to present only an example in this paper. Our example will be given in the order sketched by figure 6. We

will first sketch some requirements and then discuss partial models that should result from analysis and design (see complex activity “modeling” in the activity diagram above). The next complex activity “code generation” introduces code generation support of IOP. The final complex activity “coding and configuration” hints on necessary hand-coding and system configuration.

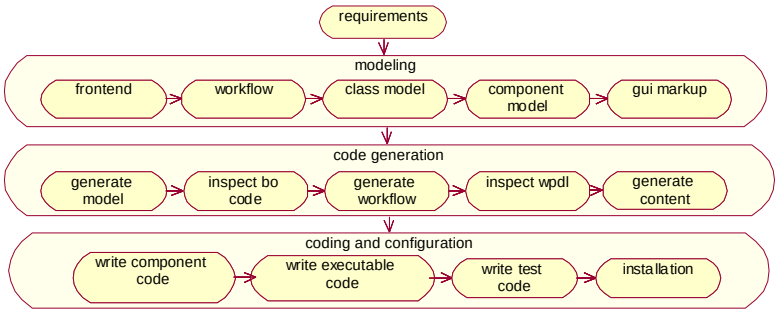


figure 6: Recipe for Example Application

5.1 Requirements

Our example application is named “WikiCms”. It shall combine automatic linking features and ease of use of the wiki brainstorming tool with content management features. Following are the requirements (incomplete, but sufficient for our example):

- web front-end based on html
- complete import of existing file systems
- automatic content linking based on file system folders and subfolders

5.2 Partial Models

Let us start with a front-end page for importing a file system (see figure 7). It is a standard html page where you enter the source path of an import folder and a target mount point.

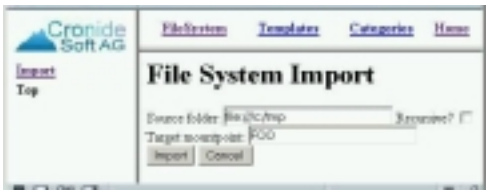


figure 7: Import Page

The corresponding workflow has four activities (see figure 8). Activity “Import_Start” imports the source folder and instantiates meta nodes in main memory. Then it (xor-) splits to either activity “Import_Save” (saving the object graph in the component) or “Import_Error” (analyzing the error cause) depending on the workflow relevant data named “successful” of type boolean. Both

activities (xor-) join again into activity “Import_End” that triggers the next front-end page to show up (see exit-action). Each activity is defined as “IMPLEMENTATION” of type “Tool” (see do-actions).

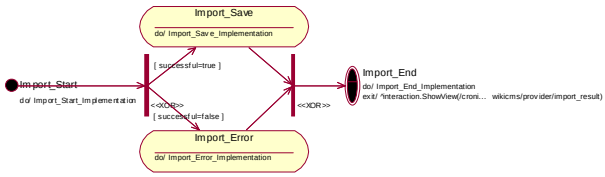


figure 8: Activity Diagram

The class model contains a class diagram on meta directory objects (see figure 9). Basically, we need the business objects “Meta Node” (folders and files), “Meta Directory” (subclass for folders), and “Meta File” (subclass for files). The objects are managed by the Meta Directory Component (component model not shown).

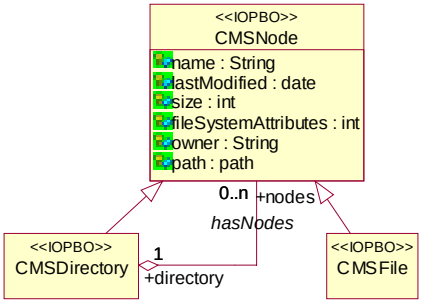


figure 9: Class Diagram

The page “top.htm” (see code below) is the start page for navigating through the imported file system. It is specified in html plus the IOP tag library (close to JSP). First, you initialize the model part of the model-view-controller pattern (MVC) by specifying a business object (“iop:useBean”) to use for data exchange between the activity (the controller part) associated to the page (the view part). Then, you specify access paths (“Directory.ObjectId”). A path can, e. g., be used for linking an activity to show up a node info page (see “action” inside “iop:a”) to the name of a folder (see “iop:value-of”). The “iop:for-each”-tag then loops through the collection of nodes under a directory.

```
<!--top.htm-->
...
<iop:useBean id="Directory"
  class="cronides...CMSDirectory" >
  ...

  <iop:input type="hidden"
    name="Directory.ObjectId"
    value="Directory.ObjectId"/>

  <iop:a href="action=shownodeinfo&amp;
    objectId={Directory.ObjectId}">
    <iop:value-of value="Directory.Name"/>
  </iop:a>
  ...

  <iop:for-each select="Directory.Nodes"
    alias="nodes" >
    ...
  </iop:for-each>
...
```

5.3 Code Generation

First, one compiles the UML model. The result is a nested folder or package structure containing generated Java sources (let us skip generation of XML and SQL). Furthermore, object type system and design component are updated. Now, we inspect the gener-

ated Java code. The example shows part of “CMSFile.java”. Amongst other snippets a package statement, an import section, the class definition (see “class”), a serial number (for mapping design entries to Java code snippets), constructors, and inner interfaces have been generated. All other generated classes (not shown here) for the business object deliver default implementations for the inner interfaces.

```
package ...component.metadirectory;
import  ...

public final class CMSFile
  extends IOPObject {
  static final long serialVersionUID = 859...;
  ...
//constructors
//inner interfaces for entity, behavior, ...
  ...
  public interface Behavior
    extends Entity, CMSNode.Behavior{}
  ...
}
```

The next step is to compile the definitions. Workflows can also be exported to and imported from WPDL and WPDL/XML (our XML-version of WPDL, upcoming XPDL proposal will be evaluated). The following code is an excerpt from the workflow model. You can see the definition of activity “Import_Start” which is of type IMPLEMENTATION and wraps the tool “Start_Implementation”. This tool is defined in the application section and links to a Java class via “ToolName”. You can also see that the activity defines an xor-split thus restricting the mentioned transitions. In case your input to the html page is correct, the workflow engine will decide to move on to the next activity “Import_Save” (or to activity “Import_Error” otherwise).

```
...
 <WorkflowProcessDefinition
   Id="WikiCMS.Import"
   Name="import" Created="2002-02-06">
   <Activity Id="Import_Start"
             Label="Import Start Node">
     <ActivityKind Type="IMPLEMENTATION">
       <ActivityImplementation>
         <GenericTool Tool=
           "Start_Implementation"/>
       </ActivityImplementation>
     </ActivityKind>
     <TransitionRestriction>
       <SplitCharacterisation Type="XOR"
         Transitions=
           "T_Start_Save T_Start_Error"/>
     </TransitionRestriction>
   </Activity>
   ...
   <Application Id="Start_Implementation"
     Label="Start Activity Implementa..."
     ToolName=
       "...CMSWfImportTraverseCode"/>
   ...
 </WorkflowProcessDefinition>
```

Next, you compile the content, that is the markup pages that make up the front-end of the application. Since IOP supports bulk-content upload your input can be a complete (web) site spanning all pages including supported file types like gif. The content compiler builds a content management structure describing the content from a logical perspective (sites, sub-sites, resources, elements, and so on). Then, it constructs a far more efficient run-time representation (compiled content) used by optimized content viewers. Content run-time also supports mapping of logical content to many physical contents by mime-type. At activity run time physical contents will be fetched according to specified target formats. In case of dynamic content the mentioned viewers assemble static and dynamic content snippets. Dynamic content snippets contain executable code for identifying and inserting business objects and their attributes.

5.4 Coding and Configuration

In order to complete the implementation stack you first implement designed components. Freed from technology issues corresponding to communication and storage you only concentrate on the low-level business logic. E. g., the meta directory component implements an operation named “search NodesByPath()” (see code). After initialization of local variables (like “query” which is in fact configured outside the component and now only identified by a name) query parameters are set (see “setPlaceHolderValue”) and the query is executed (see “execute”). The result set is then copied into a node list (see while-loop) which is returned to the caller.

```
//CMSMetaDirectoryComponent
public CMSMDNode.List searchNodesByPath
  ( String searchString )
  throws IOPComponentException
{
  ...
  IOPQuery query =
    getQuery("findNodesByPath");
  ...
  try {
    if(query != null) {
      query.setPlaceHolderValue
        (1,new,IOPPath(searchString));
      rs = query.execute( ap );
      while ( rs.hasNext() ) {
        node = (CMSMDNode.Entity)
          rs.nextObject();
        list.add(node);
      }
      rs.close();
    }
  }
  catch( IOPException e0 ) {...}
  return list;
}
```

Next, you implement executables (called by activities). The following example belongs to the executable “CMSShowFolder”. There is one public method named “execute()” (see code below). Access to components is prepared by calls to “getComponent()”. The page to be used for displaying folders is specified at the activity (using, e. g., the attribute View = ”/wikicms/top.htm”). The folder that shall be used to dynamically fill that page is simply handed over as an OID (see “objectId” and “getNode” in the code below and remember the code in the html page). The OID is read from the incoming request (user clicked a folder link before). Finally, you set the fetched node as model for the view.

```
//CMSShowFolder
public void execute()
  throws IOPWfException {
  ...
  mdc = (CMSMetaDirectoryComponent)
    getComponent( "metadirectory" );
  ...
  String oidString =
    getStringParameter("objectId");
  IOPObject.Id oid =
    IOPObjectId.valueOf(oidString);
  ...
  CMSMDNode.Entity node =
    mdc.getNode(oid);
  ...
  addViewModel("Directory", node);
  ...
}
```

To save space, we skip the testing code for unit testing, feature testing, and benchmark testing. And, we skip the installation file that contains the complete configuration of the IOP application. Basically, the structures defined earlier when discussing the topology view on IOP's architecture are instantiated as an XML file.

6 Relationship to Other Work

Based on [FHB02] we discuss some criteria for enterprise frameworks. A brief comparison of Abaxx eBusiness Suite, Interactive Objects ArcStyler, and CronideSoft IOP already shows the range in functionality of enterprise frameworks (see table below).

	Abaxx	**ArcStyler**	**IOP**
foundation	J2EE/EJB	UML, J2EE/EJB	UML, Java, XML
central aspect	assembling of multi-channel process portals	applicaton-server-specific code generation	code generation and technology encapsulation
method	-	Convergent Architecture	any method; tasks are predefined
focus of functionality	personalised content delivery, CRM	architectural IDE based on MDA	infrastructure bridging business and technology
gui support	+	--	++
protocols	+	--	++
component model	EJB	EJB, Web Services	IOP Components, EJB, CORBA
persistence mappings	--	--	++
workflow	proprietary	UML state charts	WPDL, WPDL-XML
# aspects	2 (caching , personalization)	1 (security patterns)	8 (caching, localization, logging, object type system…)
extensibility	-	+	++
# business themes	7 (access control, content management, data extr…)	0	5 (access control, content management, …)

Abaxx concentrates on ease-of-use providing a lot of tools and predefined business themes for building personalized portals. The suite fully depends on EJB, does not provide for persistence mappings, and is rather restricted in terms of extensibility (by assembling of workflows). ArcStyler comes out to be a very sophisticated IDE. It is a specialist in generating code for different EJB containers and heavily concentrates on a methodical approach based on Model-Driven Architecture. On the other side it does not provide direct support for gui, protocols, persistence mappings, nor business themes.

7 Conclusions

This paper has introduced IOP, the Internet Operating Platform. IOP is a high-end enterprise framework combining a large number of concepts, standards, implementations, and products to a synergetic whole. It is based on three major standards: UML for modeling, XML for data exchange and configuration, and Java as programming language. All other supported technologies are encapsulated for dynamic replacement or even coexistence via configuration in the areas of front-end (HTML, XML, Java), workflow (WPDL,

UML), communication (FTP, HTTP, JMS, RMI), component architecture (EJB, CORBA), and persistence (virtual memory, XML, SQL92, SQL:1999).

IOP yields the following benefits:

- *Productivity* is increased by encapsulating protocols and technologies, by visual modeling, extensive code generation, and the architecture-driven approach.
- *Quality* is increased by development process support, extensive use of design patterns, self-reproducing capabilities, and continuous self-testing due to the appliance of code generation for framework development itself.
- *Extensibility* is increased by the concepts of IOP Interaction, IOP Workflow, IOP Objects, IOP Components, and device/drivers for plugging in communication protocols, persistence mappings, component architectures, and external systems.
- *Flexibility* is increased by sticking to common open standards and by configurable interchange of drivers.

Due to its broad approach IOP can be enriched in many ways. Our near-future work will concentrate on proving IOP in more industry projects, providing configuration/modeling/design tools, further persistence mappings (MS SQL Server), further protocols (Web Services), and plug-in of specialized integration tools.

References

[Am99] Scott W. Ambler, Mapping Objects to Relational Databases, white paper, AmbySoft Inc., 1999, http://www.AmbySoft.com/mappingObjects.pdf.

[Co01] John Cowan (ed.), XML 1.1, W3C Working Draft, http://www.w3.org/TR/xml11, W3C, 2001

[Cr02] http://www.cronidesoft.com/products/iop.html, 2002

[FHB02] Mohamed E. Fayad, David S. Hamu, Davide Brugali: Enterprise Frameworks Characteristics, Criteria, and Challenges. Communications of the ACM, Vol. 43, No. 10, October 2000.

[ISO99] ISO/IEC 9075-1:1999 Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)

[OMGa] http://www.omg.org/

[OMGb] Workflow Management Facility Specification, OMG, 2000.

[RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch, The Unified Modeling Language Reference Manual, ISBN 0-201-30998, Addison-Wesley, 1999.

[Ro99] Ed Roman, Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition, ISBN 0-471-33229-1, Wiley, 1999.

[Ru01] Craig Russell (ed.), Java Data Objects, JSR000012, Version 1.0, Proposed Final Draft, Sun Microsystems Inc., 2001.

[Sc03] Stefan Schaefer, The Architecture of the Internet Operating Platform, white paper, CronideSoft AG, Germany, 2003.

[WfMC] WfMC TC-1016-P Interface 1: Process Definition Interchange - Process Model, official release, Workflow Management Coalition, 1999.

[W3C] http://www.w3c.org/

Überwachung von Aggregationszuständen in verteilten komponentenbasierten Datenproduktionssystemen

1) Anja Schanzenberger, 2) Colin Tully, 3) Dave R. Lawrence

1) GfK Marketing Services, Nordwestring 101, 90319 Nürnberg /
School of Computing Science, University of Middlesex, Bounds Green Road
London N11 2NQ, UK
2) 3) School of Computing Science, University of Middlesex, Bounds Green Road
London N11 2NQ, UK

Email: anja.schanzenberger@gfk.de, C.Tully@mdx.ac.uk, dave7@mdx.ac.uk

Abstract: Einer der signifikantesten Vorteile von Datenbanksystemen besteht in der Möglichkeit enthaltene Daten zu aggregieren und damit die Bildung beliebiger Gesamtbeträge, Statistiken oder auch Durchschnittswerte zu gewährleisten (aus vielen Datensätzen wird als Ergebnis ein Datensatz erstellt). Ebenso wichtig wie das Zusammenführen von Daten ist die Disaggregation. Darunter ist hier, im Gegensatz zur Aggregation, die Splittung von aufaggregierten Daten in ihre Einzelbestandteile zu verstehen (für den Betrachter werden aus einem Datensatz viele Ergebnisdatensätze).
In diesem Bericht werden nicht die mathematischen Möglichkeiten zur Aggregation und Disaggregation untersucht. Vielmehr zielt der Fokus auf die Nachvollziehbarkeit und damit auf die Überwachung dieser Vorgänge ab. Angenommen eine Aggregation wurde vorgenommen, wie können anschließend die eingegangenen Datensätze eindeutig wieder identifiziert und damit die Grundgesamtheit der Aggregation unmissverständlich abgeleitet werden, wenn die zu Grunde liegenden Datenbestände sich stetig ändern? Fragestellungen dieser Art werden in diesem Bericht aufgegriffen und untersucht.
An einem konkreten Beispiel aus der Wirtschaft werden die Probleme mit Aggregationszuständen erläutert. Das reale Beispielunternehmen, die GfK Marketing Services, eignet sich besonders gut für diese Fragestellungen, da ihr Kapital und ihre Geschäftsidee in der Verarbeitung und Veredelung äußerst großer Datenmengen besteht. Dabei spielt ein bald weltweit verteiltes Datenproduktionssystem in den vielen Niederlassungen des Unternehmens eine tragende Rolle. Für genau dieses komponentenbasierte verteilte Datenproduktionssystem wird demnächst ein Planungs- Control- und Monitoring System benötigt, das genaue Auskunft über enthaltene Dateninhalte im Datenproduktionssystem bieten muss. Dies erweist sich als Grundlage der Untersuchungen über die Aggregationszustände in diesem Bericht, welche daran erläutert, diskutiert und evaluiert werden sollen.

1. Einleitung

Die hier bedeutsamen Aggregationszustände entstehen nicht aus Datenmanipulation. Sie entspringen schlicht aus „GROUP-BY“-Anweisungen durch normale Abfragen. Bei derartigen Gruppierungen können wie bekannt sämtliche mathematischen Funktionalitäten einer Datenbank Verwendung finden (z.B. sum(), avg(), etc.) und damit *Aggregationen* durch *Aggregationsvorschriften* gebildet werden (siehe Abbildung 1). Beispielsweise entstehen aus mehreren Lieferperioden eine Auswertungsperiode.

Eine *Disaggregation* stellt das Gegenstück zur Aggregation dar und bedeutet deshalb die Erzeugung von Untermengen aus einem Datensatz. Von einer Auswertungsperiode sind bei der Disaggretation beispielsweise wieder die Lieferperioden abzuleiten. Aggregationszustände existieren jeweils vor beziehungsweise auch nach einer Dis-/Aggregation und sind damit eindeutig bestimmbar.

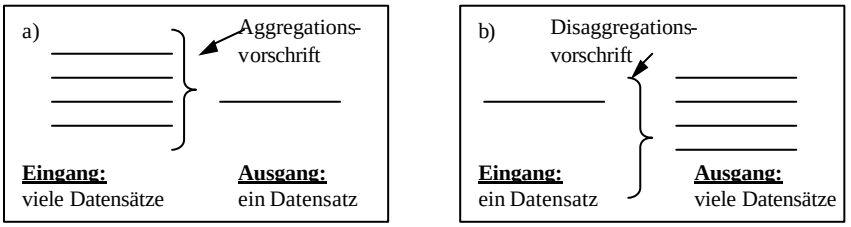


Abbildung 1: a) Aggregation b) Disaggregation

Am Beispiel der Produktionsumgebung und datenintensiven Umgebung der GfK Marketing Services [GfK02] finden die Untersuchungen zur Nachvollziehbarkeit von Aggregationszuständen statt.

Die GfK Gruppe ist eines der weltweit führenden Marktforschungsunternehmen. Deren Informationsdienstleistungen nutzen Industrie, Handel, Dienstleister und Medien für Marketing-Planungen und -Aktionen auf regionaler, nationaler sowie internationaler Ebene. Die GfK Marketing Services ist eines der vier Hauptgeschäftsfelder der GfK Gruppe. Sie hat sich auf Informations- und Beratungsservice im Bereich des Handels spezialisiert und nutzt die kontinuierliche Beobachtung von modernen technischen Gebrauchsgütern, um Berichte und Analysen über Marktsituationen zu gewinnen. Handelspanels, wie sie die GfK Marketing Services bietet, sind heute in der Lage Informationen höchster Präzision zu Verkauf, Verkaufspreis und Distribution zu liefern. Nicht nur deutschlandweit, sondern weltweit werden diese sogenannten Fakten gesammelt und zu Berichten extrapoliert und aufbereitet. Heute existieren weltweit bereits Tochtergesellschaften und Beteiligungen in über 40 Ländern.

IT-technisch gesehen ist die weltweite Verteilung der Niederlassungen eine große Herausforderung (siehe [Li01], [LH00]) für das in Deutschland bereits weitgehend existierende komponentenbasierte Datenproduktionssystem. Es handelt sich bei diesem um ein System, mit dem in unserem Fall Berichtsdaten gesammelt, veredelt und extrapoliert werden, im Gegensatz zu einem normalen Produktionssystem, welches zur Herstellung von güterorientierten Produkten, wie beispielsweise Autos, dient.

Es ist zu großen Teilen basierend auf COM-Technologie [Pa00] realisiert und wird stets weiterentwickelt. Produktionstechnisch werden die Daten von den lokalen Niederlassungen in die zentrale Niederlassung (Deutschland-Nürnberg) übertragen, dort gesammelt und schließlich zu Kundenberichten hochgerechnet. Dabei ist eine gemeinsame organisatorische Aufteilung der Produktionslandschaft, wie sie in Abbildung 2 zu finden ist, sehr hilfreich.

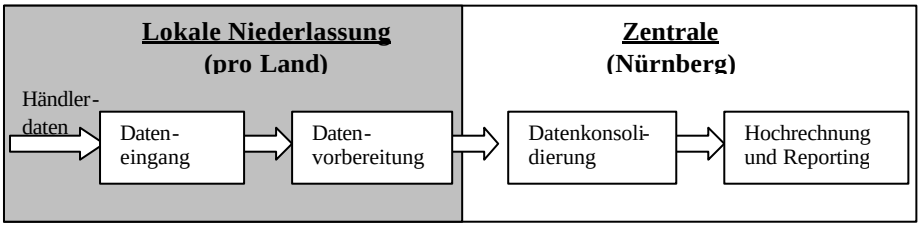


Abbildung 2: Organisatorische Sicht der Daten-Produktionsstrecke

Der Produktionsprozess beginnt mit dem (elektronischen) Einlesen der Daten beim sogenannten Dateneingang. Dieser lokale Bereich jeder Niederlassung ist zusätzlich für eine einheitliche Formatierung der Händlerdaten zuständig. Anschließend erfolgt durch die ebenfalls lokal befindliche Datenvorbereitung eine Zuordnung (Identifikation) von noch nicht (sonst automatisch) identifizierten Händlerartikeln zu einem Gfk-Artikelstamm. Alle identifizierten Händlerdaten fließen danach zur zentralen Datenkonsolidierung. Abschließend können nun die Extrapolation und das Reporting der Daten erfolgen.

Eine Herausforderung in naher Zukunft und Gegenstand der Diskussion in diesem Beitrag ist ein beabsichtigtes Planungs-, Control- und Monitoring System (PCMS) (für eine detaillierte Beschreibung sei der Leser auf [LS02] und [STL02] verwiesen), mit dessen Hilfe nicht nur die einzelnen Prozessschritte koordiniert, sondern auch eine systemtechnische Überwachung der beteiligten Produktionskomponenten erfolgen soll[1]. Als Besonderheit wird in diesem PCMS eine kontinuierliche inhaltliche Prüfung der Daten bzw. der dadurch repräsentierten Informationen (in unserem Fall: Verkaufszahlen) adressiert. Insbesondere die Problematik der permanenten Datentransformation (z.B. Aggregation und Splitten von Daten, aber auch Veränderung der Primärschlüssel) ist ein Grund dafür, dass bisher bestehende kommerzielle Werkzeuge zur Geschäftsprozessmodellierung (gemeint sind Workflow-Managementsysteme) sich nur bedingt für den Einsatz eignen und eine Eigenentwicklung auf einer soliden technischen Basis erzwungen wird.

Ziel und Inhalt dieses Beitrags ist es, Teile des nun grob skizzierten PCMS der GfK Marketing Services in Bezug auf die Nachvollziehbarkeit und Überwachung von Aggregationszuständen in derartigen datenintensiven Umgebungen zu untersuchen und konkrete Ansätze an diesem Beispiel zu diskutieren.

[1] Mit dem Teilbegriff „Control“ ist in diesem Zusammenhang eine Überwachung und nicht finanztechnisches Controlling gemeint.

2. Das Datenproduktionssystem der GfK Marketing Services

Um die Untersuchung von Aggregationszuständen für das geplanten PCMS definieren und motivieren zu können, ist es wichtig, einige Prozessabläufe des Datenproduktionssystems zu erläutern.

Jede Datenlieferung eines Händlers (bestehend aus einer Anzahl von verkauften Artikeln aus einer bestimmten Beobachtungsperiode) an den Dateneingang wird als Job bezeichnet. Jeder Job muss dabei das Produktionssystem durchlaufen. Alle Händlerartikel werden während des Prozesses entsprechenden GfK-Pendants (sog. GfK-Stammartikel) zugeordnet und weiteren Kategorisierungen wie z.B. Produktwarengruppen (Fernseher, Geschirrspüler, etc.) unterzogen. Dieses Ereignis wird als Identifikationsvorgang bezeichnet. Ein unbekannter Artikel wird dabei manuell durch Personal identifiziert, also einem sogenannten GfK-Stammartikel zugewiesen. Treffen nun in dieser oder auch in anderen Datenlieferungen weitere Artikel der Ausprägung des ersten Artikels ein, so repräsentiert der erste Artikel alle weiteren gleichartigen. Dies hat zur Folge, dass alle weiteren gleichartigen Artikel sofort automatisch erkannt und weitergeleitet werden, ohne dass erneut manuelles Eingreifen erforderlich ist.

Logische Objekte am Dateneingang (z.B. Datenlieferungen) stehen zum Ergebnis am Ende der Produktionskette (z.B. Produktwarengruppen) dabei in einer N:M-Beziehung. Ähnlich verhält es sich auf der zeitlichen Dimension, da aus ursprünglichen Datenlieferungen, die gemäß einer individuellen Lieferperiodizität (z.B. wöchentlich, monatlich, 2-monatlich) in das System eingespeist werden, Auswertungsperioden (z.B. wöchentlich, monatlich, ½-jährlich) für die Kundenberichte entstehen.

Besonderheiten:

Wie in [LS02] bereits ausführlicher dargestellt, unterscheidet sich das Datenproduktionssystem der GfK von klassischen Produktionssystemen. Beispielsweise wird eine Stichprobenerhebung von Händlerdaten vorgenommen, um die Marktereignisse wiederzuspiegeln. Ebenso kann es ausreichen, nur gewisse Prozentsätze der eingegangenen Daten auszuwerten, da durch statistische Hochrechnungen bereits aussagekräftige Ergebnisse erzielt werden können. Die für die Auswirkung auf die Aggregationszustände signifikanten Eigenschaften werden im folgenden noch einmal kurz vorgestellt.

- *Statische, jedoch konfigurierbare Produktionsabläufe*
  Bei der Produktion von Daten für die Erzeugung von Kundenberichten ist im wesentlichen ein fest vorgegebenes Prozessschema zu verfolgen, sodass eine flexible Entscheidung durch Evaluierung komplexer Bedingungen durch das PCMS nicht notwendig ist.
- *Daten als Produkt*
  Im Unterschied zur klassischen Produktion von Gütern kann es in dem vorliegenden Szenario vollkommen ausreichen, wenn der Sollzustand des Endprodukts nur zu einem bestimmten Anteil (z.B. 80%) erreicht ist. Die Erlangung von 100% des Sollzustandes würde bedeuten, dass alle Daten die in das Endprodukt „Bericht“ einfließen vollständig durch das gesamte Datenproduktionssystem gelaufen sind.

Jedoch bereits in einem unvollständigeren Zustand sind am Ende der Produktionskette ein Reporting zu realisieren und Trends zu erkennen. Dies beruht auch auf dem statistischen Stichproben-Prinzip.

- *Änderung der logischen Objekte*
  Datentransformation bedeutet, dass eingehende Objekte nicht mehr eindeutig produzierten Objekten zugeordnet sein können. Vielmehr existiert eine N:M-Beziehung zwischen Eingangs- und Ausgangsdaten. Im Gegensatz dazu sind in einer klassischen Produktionskette die Einzelteile im Endprodukt stets identifizierbar.

Die inhaltliche bidirektionale Abbildung der verwendeten Terminologien zwischen Anfang (Dateneingang) und Ende (Reporting) ist eine der wesentlichen Anforderungen an das PCMS. Konkret heißt dies, die Veränderung der Primärschlüssel zieht sich durch das ganze System und muss in jedem Fall nachvollzogen werden können. Die Veränderung von Lieferungen zu Auswertungen lassen somit keine 1:1 Beziehung zu.

3. Das Planungs- Control- und Monitoring System der GfK Marketing Services

Das PCMS setzt auf die Prozessabläufe des Datenproduktionssystems auf. Zum einen ist das Ziel die *Kontrolle* über den eben beschriebenen laufenden Betrieb zu übernehmen. Dabei müssen Auskünfte über den momentanen Zustand der Aufträge ebenso eingerichtet werden, wie schnelle Identifikation und Reaktion in Fehlersituationen. Zum anderen ermöglicht der Einsatz des PCMS eine *Optimierung* der Business Prozesse hin zu einer bedarfsgesteuerten Produktion. „Durch eine bedarfsgerechte Planung und Steuerung auf Anwendungsebene soll eine berichtsorientierte Produktion („*pull-based production*“) im Gegensatz zu der bisherigen lieferungsorientierten Produktion („*pushed-based production*“) stattfinden“ (siehe [LS02]). Eine gleichmäßigere Personalauslastung wird durch eine konsequente Verfolgung und Priorisierung der Aufträge erreicht werden. Daraus ergeben sich schließlich die nun folgenden Basisfunktionalitäten für das PCMS.

Basis-Aspekte des PCMS:

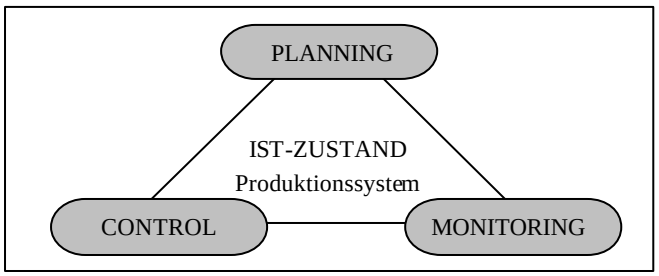


Abbildung 3: Basis-Funktionalitäten des PCMS

Die drei Eckpfeiler des PCMS (siehe Abbildung 3) bestehen im wesentlichen aus einer *Planungskomponente*, die einen *SOLL-Zustand* repräsentieren wird. Durch historischen Vergleich von Kennzahlen (z.B. Verkaufszahlen aus Vorperioden) soll für die gesamte Prozesskette ein TQM (Total Quality Management) stattfinden können. (Gute Planungsansätze können in [MG01] gefunden werden). Der zweite Eckpfeiler ist das *Monitoring.* Durch Monitoring wird der *IST-Zustand* im Produktionssystem abgebildet. Sowohl eine systemtechnische Überwachung der Hardware, als auch eine Überwachung der Jobs auf Anwendungsebene soll einen reibungslosen Prozessverlauf gewährleisten. Darüber müssen damit genaue Aussagen über die jeweiligen Bearbeitungszustände der einzelnen Aufträge erzielbar sein. Die Realisierung wird über ein Logging der beteiligten Komponenten stattfinden. Der dritte und letzte Eckpfeiler ist der *Kontrollteil* des PCMS. Die Möglichkeit steuernd auf den IST-Zustand im Datenproduktionssystem einzugreifen wird einerseits durch eine Vergabe von Job-Prioritäten erreicht. Andererseits werden auf systemtechnischer Ebene lokale Eingriffe zur Hardware und Software Überwachung eingerichtet.

Aspekte zur Überwachung von Aggregationszuständen

Durch die Basis-Funktionalitäten wird den PCMS-Anwendern bereits ein umfassendes und sehr wichtiges Werkzeug zur pünktlichen Erstellung der Kundenberichte an die Hand gegeben. Jedoch wird ein weiterer wichtiger Anspruch an das PCMS gestellt, der genau auf das interessierende Thema der Aggregationszustände zielt.
Zur Verifikation einzelner Kennzahlen in Kundenberichten, aber gerade auch bereits in vorherigen Stadien des Prozesses werden zwei weitere wichtige Bestandteile des PCMS gefordert.

1. Summenprotokolle (Summaries)

Am vorteilhaftesten nach jedem beliebigen Prozessschritt, aber mindestens nach bedeutenden Prozessabschnitten, werden Zusammenfassungen der Verkaufszahlen in Form von Aufaddierung der Werte innerhalb eines Jobs gefordert. Damit können Aussagen über die Richtigkeit und Vollständigkeit des Dateninhaltes getroffen werden. Dies stellt eine weitere wichtige TQM Maßnahme dar, mit der frühzeitig bei Fehlentwicklungen ein Eingriff in den Prozess stattfinden kann (z.B. auch durch Stoppen und völlig neu Starten eines kompletten Jobs). Zusätzlich wird eine historische Speicherung der Summenprotokolle vorgenommen, um Vergleichswerte für sich periodisch wiederholende Jobs zu erhalten.

2. Einzelsatz-Tracking

Zwei elementare Ziele sollen mit dem sogenannten *Einzelsatz-Tracking* verfolgt werden:

a) Nachverfolgung einzelner Händler-Artikel bis zum Kundenbericht.

Zur Evaluierung und Verifikation der gefundenen Zahlen in den Kundenberichten (bzw. in jedem vorherigen Prozessschritt) wird eine Methode benötigt, mit der einzelne Artikel vom Beginn des Prozesses bis zum Ende genau nachverfolgt und überwacht werden können (siehe Abbildung 4).

Auch in Fehlersituationen soll die Nachverfolgung eindeutige Aussagen bereitstellen können, *ob* und in *welcher Weise* ein Artikel an einer Komponente verarbeitet wurde und in welche Datenströme er nach der Komponente mit eingeflossen ist.

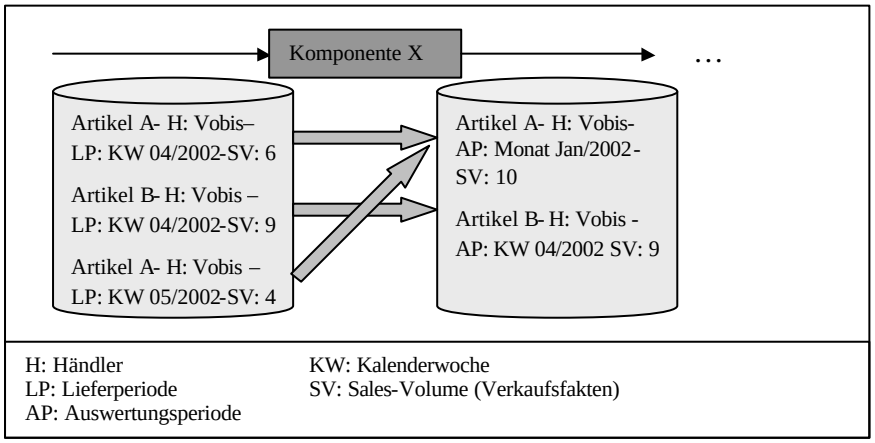


Abbildung 4: Beispiel zur Nachverfolgung der Aggregation von Lieferperioden zu Auswertungsperioden an einer Komponente

b) Simulation von geplanten Produktionszyklen

Weniger brisant, aber sicherlich dennoch ein möglicher Bestandteil des PCMS, könnte eine Simulation konkreter Datensätze durch das Datenproduktionssystem darstellen. Simuliert würde dabei der gesamte Prozessablauf, jedoch ohne das die Ergebnisse der Simulation die tatsächliche Produktion beeinflussen. Die Komponenten führen wie bei der echten Produktion die einzelnen Bearbeitungsschritte durch (Simulationsdurchlauf mit möglichst kleinen Datenmengen und geringer Priorität, damit die echte Produktion zeitlich nicht zu sehr beeinträchtigt wird). Auf Basis der ohnehin benötigten konkreten Systemkennzahlen (z.B. maximale Durchlaufzeit von Jobs an einer Komponente) und der erstellten Erfahrungswerte (z.B. Summenprotokolle) könnte eine Simulation ganzer Jobs und vielleicht auch einzelner Datensätze geschaffen werden, um *„Was-wäre-wenn"-Szenarios* mit in die Betrachtungen einbeziehen zu können. Beispielsweise ETL-Werkzeuge (siehe [BG01]) bieten derartige Funktionalitäten.

4. Strategien zur Nachvollziehbarkeit von Aggregationszuständen

Tubing System

Da, wie in Kapitel 2 bereits vorgestellt, ein relativ statischer Produktionsablauf im Datenproduktionssystem die Basis darstellt, erscheint für die Realisation des Einzelsatz-Trackings das sogenannte *„Tubing System"* sinnvoll.

Als Beispiel möchte ein Benutzer des PCMS nachvollziehen, wie bzw. ob ein bestimmter Artikel in das Endergebnis „Kundenbericht Y“ eingeht. Dafür gibt er die Identifikationsparameter für diesen bestimmten Artikel in das System ein. Dadurch wird nun der Tubing-Mechanismus ausgelöst. Nach der Reihe wird nun jede Komponente im statischen Workflow besucht und der Datenfluss für diesen Artikel nachvollzogen. Das Tubing-System funktioniert für die folgenden beiden Fälle und deckt damit ausreichend alle gewünschten Überwachungssituationen ab:

a) Kompletter Workflow-Durchlauf

Entwicklungen wie und ob ein bestimmter Artikel sich durch das Datenproduktionssystem seinen Weg gebahnt hat, werden rekonstruktabel. Während des Prozesses finden immer wieder Qualitätssicherungsmaßnahmen statt. Fachexperten haben erstmalig die Chance fehlerhafte Daten (z.B. vom Händler falsch geliefert etc.) und widerspruchsvolle Berechnungen (z.B. fehlerhafte Behandlung negativer Verkaufszahlen etc.) zu identifizieren. Stichprobenartig können Durchläufe zur Überwachung der Vollständigkeit von Kundenberichten gestartet werden. Durch das Tubing System können die Ergebnisse der Nachverfolgung an jeder einzelnen „Station“ (Komponente) angezeigt werden und somit ein übersichtliches Gesamtbild der Produktionskette im Bezug auf den gesuchten Artikel dargestellt werden.

b) Erkennen von Fehlersituationen durch Workflow-Testdurchlauf

An jeder Komponente werden Fehler geloggt. Dabei können zwei Fehlersituationen entstehen. Die erste Möglichkeit ist, eine Komponente loggt einen Fehler, kann aber trotzdem weiterarbeiten. Die zweite Situation entsteht, wenn der aufgetretene Fehler so schwerwiegend ist, dass die Komponente ihre Bearbeitung abbrechen muss. In beiden Fällen kann eine Überprüfung einzelner Artikel notwendig werden. Durch das Tubing System kann genau nachverfolgt werden, ob und wieweit ein bestimmter Artikel noch richtig verarbeitet wurde. Dabei wird wie bei der vollständigen Nachverfolgung der Artikel vom Eingang des Systems bis zu der zu überprüfenden Komponente bzw. einer Nachfolgerkomponente durchgeschleust. Aus Performanzgründen ist es vorteilhaft, dass nicht der komplette Workflow durchsucht werden muss.

Eigenschaften der Überwachung

Das Nachvollziehen der Aggregationszustände muss an jeder Komponente einzeln angebracht werden. Dies hängt damit zusammen, dass jede Komponente andere Aggregationsvorschriften besitzt und außerdem die Identifikationsschlüssel am Eingang einer Komponente sich zu denen am Ausgang einer Komponente wie bereits in Kapitel 2 erwähnt unterscheiden können.
Die Möglichkeiten zur Überwachung von Aggregationszuständen werden nun auf die folgenden Eigenschaften hin überprüft:

Eigenschaft	Ausprägung	Beschreibung
Datenmenge	Statisch / nicht statisch	Eine statische Datenmenge bleibt zu jeder Zeit gleich. Eine weitere Ausprägung wäre, Daten dürfen hinzukommen, aber nicht weggenommen werden. Ist die Datenmenge nicht statisch, darf sie sich jederzeit beliebig entwickeln.
Aggregationsvorschrift	bekannt / nicht bekannt	Die Aggregationsvorschrift beschreibt, wie die eingehenden Datenmengen aggregiert werden. Ist sie bekannt, kann sie zur Nachverfolgung herangezogen werden.
Job-Parameter	bekannt / nicht bekannt	Die Job-Parameter beschreiben, welche Datenmenge zu untersuchen ist. Job-Parameter wären in unserem Beispiel: Shop (entspr. z.B. Händler-Filiale), Produktgruppe, Lieferperiode, etc.
Speicheraufwand	minimal bis maximal	Untersuchung, welche der Überwachungsmöglichkeiten mit wie viel zusätzlichem Speicheraufwand auskommen.

Möglichkeiten zur Nachvollziehbarkeit der Aggregationszustände

Der Leser sei an dieser Stelle darauf hingewiesen, dass sich das PCMS selbst noch in der Planungsphase befindet und aufgrund dessen ebenfalls Summenprotokolle, Einzelsatz-Tracking und Tubing-System bisher nicht implementiert sind. Die im folgenden vorgestellten Ansätze basieren deshalb auf rein theoretischen Untersuchungen und wurden noch nicht durch reale Erfahrungen bestätigt.

1. ***statische Datenmengen***

   Wenn historisch gesehen zu jeder Zeit klar ist, wann welche Datenmengen zu betrachten sind, ist diese Methode die best geeignete, um Aggregationszustände vor und nach einer Komponente genau bestimmen zu können. Eine weitere Voraussetzung ist deshalb die Kenntnis über die Job-Parameter. Das Wissen über die Aggregationsvorschrift ist nicht notwendig.

   Beispiel:
   In dem kleinen Beispiel von Abbildung 5 ist die Komponente Cx dafür zuständig, aus vier Lieferperioden eine Auswertungsperiode für einen Artikel eines Händlers zu generieren.
   Angenommen Pool A ist immer pro Tag gültig. Erst nachts, wenn kein Anwender Aktionen ausführt, wird Pool A mit neuem Datenmaterial bestückt. Dann ist pro Tag Pool B immer durch die Job-Parameter reproduzierbar.

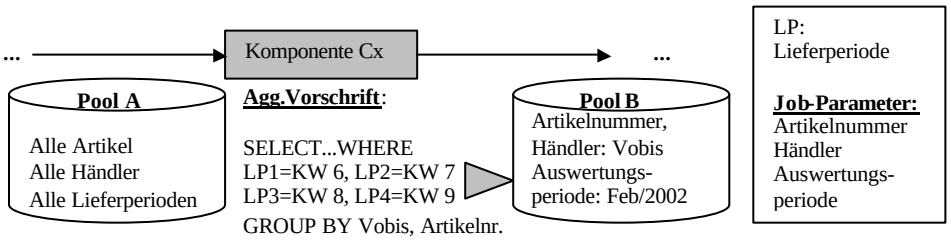


Abbildung 5: Beispiel: statische Datenmengen

Vorteil:

Es fällt kein zusätzlicher Speicheraufwand zur Überwachung an. Eine genaue Aussage kann zu jeder Zeit getroffen werden. Können dazu noch statische Datenmengen historisch gespeichert werden, wird eine schrittweise Nachverfolgung aufgrund der historisch gehaltenen Daten möglich.

Nachteil:

Für die Speicherung der historischen Datenmengen fällt jedoch meist ein zu großer zusätzlicher Speicheraufwand an. Diese Methode ist also nur bei kleinen (historisch gehaltenen) statischen Datenmengen von Bedeutung. Alle Job-Parameter müssen ebenfalls zusätzlich gehalten werden, um eindeutig die zu überwachende Datenmenge zu identifizieren. Außerdem ist bei zunehmenden Speichermengen mit einer Verlangsamung der Abläufe im zu überwachenden wie auch im überwachenden System zu rechnen, die durch den zusätzlich entstehenden Verwaltungsaufwand entsteht.

2. ***Einzelsatz-Logging***

Wird jeder eingehende Datensatz von der Komponente komplett geloggt, wären Eingang und Ausgang der Datenmengen zu jederzeit bekannt. Bei Kenntnis der Job-Parameter können die Aggregationszustände jederzeit nachvollzogen werden.

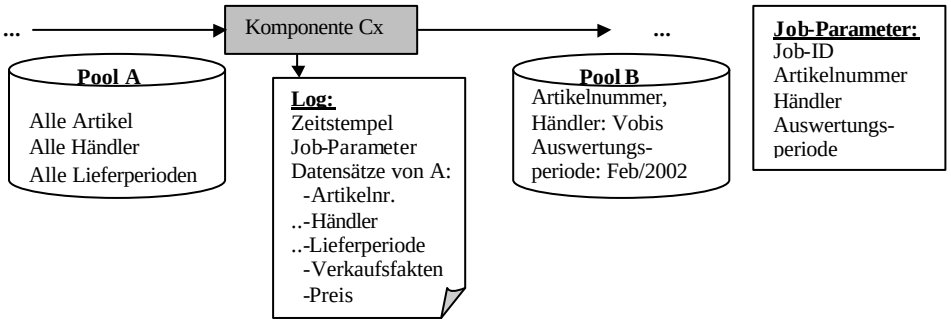


Abbildung 6: Beispiel Einzelsatz-Logging

Beispiel:

Abbildung 6 zeigt, dass alle ausgewählten Datensätze aus Pool A in ein Log

geschrieben werden. Pool B kann mit Hilfe der Job-Parameter jederzeit abgebildet werden, da die Ursprungsmenge A bekannt ist.

Vorteil:
Die Aggregationsvorschrift wird bei dieser Methode nicht benötigt, da Ein- und Ausgang der Datenmengen jederzeit zur Verfügung stehen. Auch die Datenmengen müssen nicht statisch vorgehalten werden.

Nachteil:
Es wird mindestens der doppelte Speicheraufwand für die Datenmengen benötigt. Auch hier ist deshalb eine Verlangsamung der Systeme aufgrund zusätzlichem Verwaltungs- und Speicheraufwand zu erwarten.

3. ***Primärschlüssel-Logging***

Nur die wichtigsten Primärschlüsselattribute der eingehenden Datensätze werden bei dieser Methode geloggt. Auch bei dieser Methode müssen die Job-Parameter bekannt sein.

Beispiel:
Wie bei Einzelsatz-Logging (Abbildung 6) wird geloggt. Jedoch anstatt des ganzen Datensatzes aus Pool A, werden nur dessen Primärschlüsselattribute aufgehoben (in unserem Beispiel: Artikelnummer, Händler, Lieferperiode). Ändern sich im Pool A nur die Verkaufsfakten oder der Preis eines Artikels, kann trotzdem Pool B nachvollzogen werden.

Vorteil:
Die zu speichernde Datenmenge wird im Gegensatz zu Methode 2 zumindest etwas reduziert. Beispielsweise würde in einem Einsatzfall in der GfK die Reduzierung in etwa 1/5 betragen (Verhältnis Primärschlüssel zu Verkaufsfakten).
Aggregationsvorschriften müssen nicht bekannt sein, da die Kombination aus eingehenden und ausgehenden Primärschlüsselattributen vorgehalten wird.

Nachteil:
Der größte Nachteil besteht darin, dass sich die vorhandenen Daten zu den Primärschlüsseln im Sinne von Löschungen nicht ändern dürfen. Neue Attributswerte des selben Datensatzes können hinzukommen. Jedoch dürfen keine Datensätze entfallen, da sonst ein Nachvollziehen der Ergebniswerte nicht mehr folgerichtig stattfinden könnte. Beste Voraussetzungen würden auch hier eher statische Datenmengen bieten, da sich bei diesen die vorhandenen Datensätze nicht ändern.

4. ***Datenschätzung***

Kein zusätzlicher Aufwand zur Nachverfolgung der Artikel ist bei dieser Methode erforderlich. Es werden die aktuellen Datenmengen zum Nachverfolgungszeitpunkt beim Einzelsatz-Tracking verwendet, um eine ungefähre Darstellung des Datenzustandes von dem eigentlich gewünschten echten Abarbeitungszeitpunkt zu erhalten. Das Ergebnis der Nachverfolgung an einer Komponente kann das gleiche sein, wie zum Zeitpunkt der tatsächlichen Abarbeitung. Dieser Umstand ist jedoch nicht zwingend. Ebenso kann sich die zugrunde liegende Datenmenge bereits verändert haben und damit die Nachverfolgung ein anderes (unerwünschtes) Ergebnis liefern.

Beispiel:
Ziel ist, wie in Abbildung 7 gezeigt, Pool B1 eventuell mit Verwendung von Pool A2, Job-Parametern und Aggregationsvorschrift nachzuvollziehen.
Jedoch kann sich das Abfrageergebnis (Pool B2) von Pool B1 unterscheiden, da die Abfragezeitpunkte von B1 (tatsächlicher Abarbeitungszeitpunkt) und B2 (Nachverfolgungszeitpunkt) unterschiedlich sind.

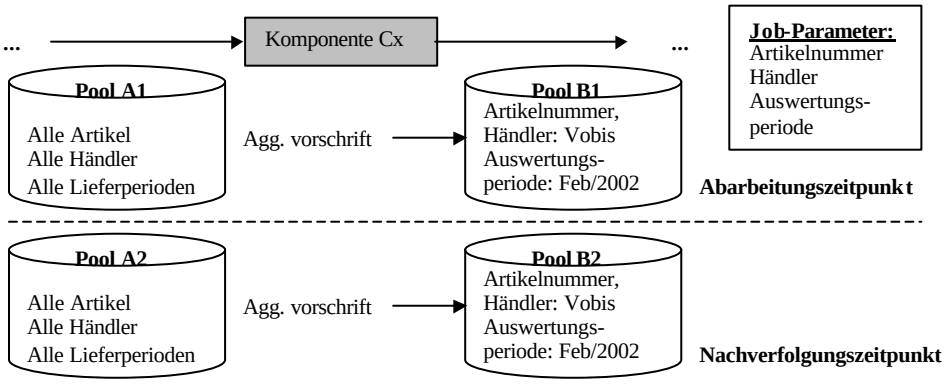


Abbildung 7: Beispiel Datenschätzung

Vorteil:
Es findet kein zusätzliches Logging statt. Deshalb wird kein zusätzlicher Speicherplatz benötigt. Auch die Datenmengen dürfen sich beliebig verändern und müssen nicht statisch gehalten werden.

Nachteil:
Die Aggregationsvorschrift und Job-Parameter müssen bekannt sein. Ersteres erzwingt einen weiteren Aufwand für jede Komponente zur Programmierung der Überwachung, um die unterschiedlichen Aggregationen nachvollziehen zu können. Der wohl größte Nachteil dieser Methode liegt darin, dass nur eine ungenaue Schätzung ermöglicht wird. Die augenblickliche Abbildung der Datenmengen kann jederzeit nachvollzogen werden, jedoch muss dies nicht dem Szenario entsprechen, mit dem z.B. ein Kundenbericht angefertigt wurde. Zusätzlich muss die zur Nachvollziehung benötigte erneute Ausführungszeit der Komponente in Laufzeitbetrachtungen berücksichtigt werden.

5. Resümee

Im Hinblick auf das Beispielszenario kann das folgende Ergebnis abgeleitet werden. Für die immens datenintensive Produktion der GfK MS wird die Methode (1) der statischen Datenmengen nicht zum tragenden Einsatz kommen können. Zu überprüfen gilt noch, ob an bestimmten Schnittstellen (lokales Land zu Zentrale bzw. Konsolidierung zu Reporting) im Produktionsablauf diese Methode trotzdem eingesetzt werden kann, da sie mit geringstem Logging-Aufwand die besten Ergebnisse für die Hauptaufgabe der Nachvollziehbarkeit erzielt.

In Umgebungen mit (evtl. historischen) statischen Datenmengen würde damit eine Empfehlung einer Strategie zur Nachvollziehbarkeit klar zu Gunsten der Methode (1) ausfallen.

Eine Anwendung des Einzelsatz-Loggings (Methode 2) im Überwachungsszenario der GfK MS ist aufgrund der datenintensiven Umgebung nicht zu erwarten. Ein Einsatz würde sich nur dann in einem Szenario rechtfertigen, wenn ein mindestens verdoppelter Speicheraufwand und eine eventuelle Verlangsamung der Systeme tolerierbar sind. Datenbanksysteme mit geringer Mengenlast und viel zusätzlich freiem Speicherpotential stellen eventuell sinnvolle Einsatzumgebungen dar. Außerdem könnte Einzelsatz-Logging sich eignen, wenn eine große Differenz zwischen vorhandenen Daten (große Menge) und Daten zur Auswertung (geringe Menge) verwendet wird. Beispielsweise in Ad-hoc Studien, in denen viele Daten gesammelt aber nur selten und wenige ausgewertet werden, könnte eine Verwendung in Betracht gezogen werden.

Das Problem für den Einsatz der dritten Methode (Primärschlüssel-Logging) ist der extrem wechselhafte Zustand der Daten im Produktionssystem. Zu keiner Zeit sind die Datenmengen in der GfK MS statisch. Zusätzlich können jederzeit Löschungen stattfinden. Ein Einsatz ist damit nur in Umgebungen zu empfehlen, die wenig Manipulationslastig sind und beispielsweise auf Löschungen ganz verzichten können. Auch ist der Aufwand an zusätzlicher Informationsspeicherung vor einem Einsatz zu überdenken. Aus diesen Gründen ist auch Primärschlüssel-Logging nicht optimal in unserem Beispielszenario.

Die Datenschätzung (Methode 4) stellt im Augenblick die wohl systemverträglichste Methode dar, mit der überhaupt (und ohne großen Aufwand) ein Einzelsatz-Tracking in der GfK MS anzudenken ist. Nach der Erstellung eines Prototyps für das Einzelsatz-Logging wird sicherlich auch klar werden, ob der Nachteil der Ungenauigkeit hingenommen werden kann, oder ob trotzdem andere Methoden in Betracht gezogen werden müssen, um eine hinreichende Überwachung des Datenproduktionssystems bereitstellen zu können. Der Einsatz lohnt sich also in Systemumgebungen in denen Ungenauigkeiten im Überwachungsergebnis akzeptierbar sind. Diese Methode besticht außerdem durch den im Vergleich zu den anderen Methoden geringsten Implementierungs- und Speicheraufwand. Anzumerken ist, dass dennoch die Systemlast erhöht wird und deshalb Einsatzfelder zu bevorzugen sind, in denen Zeiten mit geringerer Last (z.B. Nachts) zu vermerken sind.

In diesem Bericht wird deutlich, dass Aggregationen und Disaggregationen selbst wichtige Datenbankbestandteile darstellen, die ausgereifte und unverzichtbare Maßnahmen für datenintensive Businessprozesse darstellen. Wie gezeigt wurde, gestaltet sich jedoch die manchmal unverzichtbare Überwachung der dabei entstehenden Aggregatszustände als schwierig bis unmöglich.
Eine *ideale Methode* wurde dafür noch nicht gefunden, da wie gezeigt am Beispielszenario der GfK MS augenblicklich nur die Methode der ungenauen Daten-schätzung in Betracht gezogen wird. Bei dieser kann es im Einzelfall passieren, das kein befriedigendes Ergebnis gefunden wird. Das bedeutet, ein früher vorliegender Sachverhalt kann nicht detailliert überwacht bzw. nachvollzogen werden. Datensätze können hinzugekommen oder weggefallen sein. Da in den meisten Szenarios keine statischen Datenmengen vorliegen werden, kann die Datenschätzung unzureichend sein.

Eine ***datenbankseitige Unterstützung*** derartiger Überwachungsvorgänge würde einen ***deutlichen Mehrwert*** in Überwachungsfunktionalitäten darstellen, induziert aber auch gleichzeitig einen sehr hohen Anspruch an Verwaltungsinitiativen und Datenhaltung innerhalb von Datenbanken selbst.

Literaturverzeichnis

[BG01] Andreas Bauer, Holger Günzel (Hrsg.), 2001, 'Data Warehouse Systeme, Architektur, Entwicklung, Anwendung', Heidelberg: dpunkt.verlag GmbH.

[GfK02] GfK Marketing Services, 2002, [Online], http://www.gfkms.com, [2002, Sep., 02].

[LH00] Claudia Linnhoff-Popien, Heinz-Gerd Hegering (Eds.), 2000, Trends in Distributed Systems: Towards a Universal Service Market, Third International IFIP/GI Working Conference, USM 2000, Preface, Sept. 2000, Lecture Notes of Computing Science 1890, Springer Verlag, Heidelberg.

[Li01] David S. Linthicum, 2001, 'B2B Application Integration, e-Business-Enable Your Enterprise', Addison-Wesley.

[LS02] Wolfgang Lehner, Anja Schanzenberger, 2002, 'Einsatz von WebServices in datenintensiven Umgebungen', 1. Workshop von Anwendungen auf der Basis der XML Web-Service Technologie, Gesellschaft für Informatik, Arbeitskreis „Modellierung und Spezifikation von Web-Service basierten Anwendungen", FG 2.5.2.

[MG01] Peter Mertens, J. Griese: 'Integrierte Informationsverarbeitung 2'. Wiesbaden: Betriebswirtschaftlicher Verlag Dr. Th. Gabler, 2001

[Pa00] Ted Pattison, 2000, Programming Distributed Applications with COM+ and Visual Basic 6.0, 2nd edn, Microsoft Press, Redmond, Washingtion 98052-6399

[STL02] Anja Schanzenberger, Colin Tully, Dave R. Lawrence, 2002, 'A web service based approach to monitor and control a distributed component execution environment', Working Paper University of Middlesex, London, Available per email: anja.schanzenberger@gfk.de, GfK Marketing Services, Nordwestring 101, 90319 Nürnberg, Germany

Information Integration in a Global Enterprise
Some Experiences from a Financial Services Company

Robert Marti

Swiss Re
Mythenquai 50/60
CH-8022 Zurich, Switzerland
robert_marti@swissre.com

Abstract: In most commercial enterprises, information is scattered across a large number of (legacy) data stores. Moreover, it is nearly impossible to obtain funding to replace these data stores with a single integrated database, given tight budgets, time-to-market pressures and past failures of various kinds of large-scale efforts. In this paper, we advocate an approach which hinges on (1) a high-level enterprise information architecture, (2) the identification of the stable elements in this information architecture – essentially identifiers of some core entity types, the sets of legal values of some descriptive attributes – and (3) a central metadata and reference data repository which supports the tracking of the history of reference data.

1 The Application Area: Reinsurance

A reinsurance company, or *reinsurer*, insures insurance companies, also called *(primary) insurers*. Much like insurers assume risk for a fixed price from their clients, either persons or organizations, reinsurers assume risk from primary insurers. Depending on the their needs, primary insurers mainly buy reinsurance to either assume a larger portfolio of similar small risks, and/or to pass on parts of a specific very large risk. For further information on the subject of reinsurance, we refer the reader to [CL00].

While both insurers and reinsurers assume risk, they are different in that a primary insurer is a retailer with a relatively large customer base and relatively few highly standardized products, while a reinsurer is a wholesaler with a relatively small customer base and relatively large number of products specifically designed for one client. As a result, the volume of data and transactions is not nearly as much of a challenge for a reinsurer as it is for a primary insurer, or a retail bank, for that matter. On the other hand, given product diversity, the size (and therefore clout) of many clients, and the global reach of the business, standardization is much more difficult on the data side as well.

2 The Challenge: Legacy and Cost/Time Constraints

In the area of information management and databases, two major differences between working in almost any commercial enterprise as opposed to academia are:

- *Historical baggage*
  Most large, global enterprises look back over a history of legacy systems and associated data stores, often developed in isolation, without adequate documentation, using yesterday's methodologies and technology. Many of these applications still form a vital part of the operational backbone of the business and are as such highly valuable. In other words, they represent a huge investment which cannot easily be replaced.
- *Pressures with respect to cost/benefit and time-to-market*
  In today's corporate environment, obtaining funding for large, long-term projects is increasingly difficult due to cost and time-to-market pressures. Often, the best chance is to find an in-house client with budget and then catering to his specific needs, which in turn may conflict with the interests of the corporation as a whole.

3 Application Landscapes

In order to identify both gaps and (potential) overlaps in applications fielded in many different locations and supporting various reinsurance products, Swiss Re has been using a simple matrix called an *application landscape* for over 5 years:

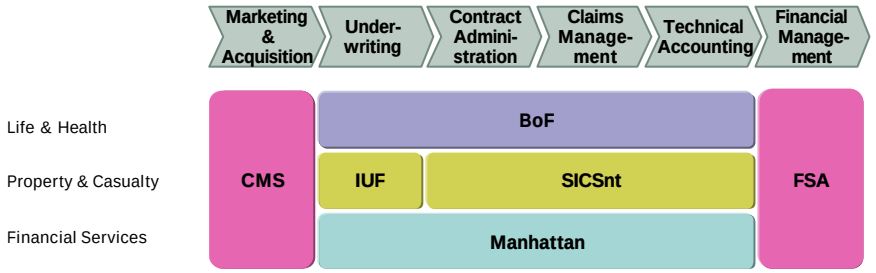


Figure 1: (Simplified) Application Landscape

The column headings represent a high-level value chain which is roughly structured into a sequence of core business processes (e.g., “Marketing & Acquisition”, “Underwriting”), while the rows are typically associated with either locations, organizational units, products, or a combination thereof (e.g., “Life & Health”, “Property & Casualty”). If an application supports one or more business processes by capturing the data which is produced during the execution of those processes, then the application is depicted underneath those processes. For example, the application CMS (Client Management System) supports data capture in the “Marketing & Acquisition” process across the entire group,

while SICSnt supports data capture of the processes “Contract Administration”, “Claims Management”, and “Technical Accounting”.

Note that drawing an application landscape is not an exact science in that

- the business processes of the value chain do not necessarily occur in the sequence shown
- the boundaries of what an application supports exactly is sometimes a little bit “fuzzy”.

In the context of the Framework for Information Systems Architecture pioneered by John Zachman [Za87, SH92, Co96, Ha03][1], application landscapes are on the level of the business owner and essentially relate activities with locations (represented by applications).

4 Architectural Framework and Piecemeal Development

Application landscapes have proven to be a useful tool for communication to the business side and top management, as well as to align development plans of different organizational units. Moreover, application landscapes highlight the need for application integration:

While the need for integrating the various information islands both along the value chain (see Figure 2) as well as across product lines and/or organizational units (see Figure 3) is unquestioned, it is nearly impossible to make a sound business case for “big bang” approaches such as developing (detailed) Enterprise Data Models or even a single integrated Enterprise Data Warehouse.

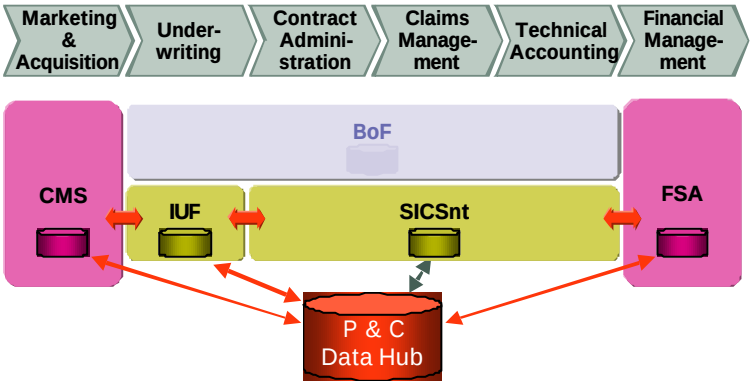


Figure 2: Integration along the Value Chain

[1] Following [Ha03], the Framework for Information Systems Architecture is a matrix with 5 layers addressing the different views of various stakeholders (planner, business owner, architect, designer, builder) and 6 columns addressing the different aspects of an information system (data, activities, locations, people, time, motivation).

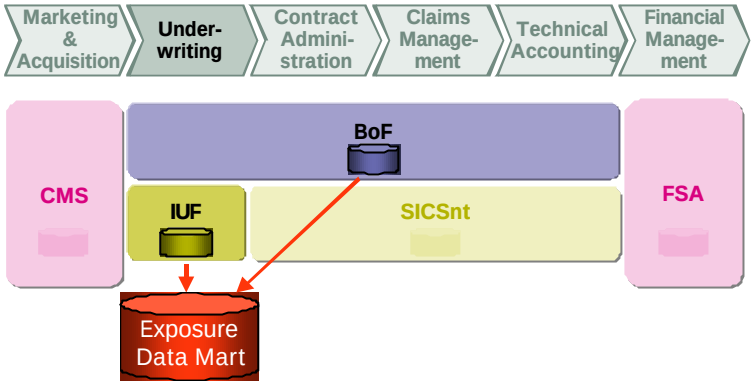


Figure 3: Integration across Organizational Units / Products / Location

A compromise is to adopt an approach which hinges on the following two elements:

- *A High-Level Architectural Framework*
  Instead of developing complete and detailed data and business process models in multi-year projects, enterprise-wide models are only elaborated as far as needed in order to get a rough overall picture, stopping at approximately a dozen subject areas (high-level entity types) and a dozen core business processes. In the context of the Zachman Framework, data and business process models are elaborated to serve the business owner's and the business architect's needs, but not down to a (system) designer's view.
- *Piecemeal development and integration*
  New solutions are developed within the confines of the high-level blueprint, based on a sound business case, in a piecemeal fashion. Details are added to the high-level models on a "just-in-time" basis, which in rare cases may even lead to amendments in existing parts of the models.

5 High Level Data Model

In order to establish a high-level architecture which allows subsequent information integration with a minimum of disruption to the existing systems and databases, the first issue is to identify the stable elements of the overall data model.

Swiss Re's high-level data model was developed in a two-pronged approach:

- Published generic and industry-specific data models, see e.g. [Ha96, Fo97, Si01], and architectures, IBM's Insurance Application Architecture (IAA, see [IB02]) and a similar application architecture developed within the German insurance industry (VAA, see [GD96]) were studied.
- At the same time, the data models of existing systems as well as the contents of various reports were analyzed in a bottom up way.

A simplified version of the basic data model, representing the business owner's view in Zachman's framework [Ha03], was established in late 1997 and looks as follows:

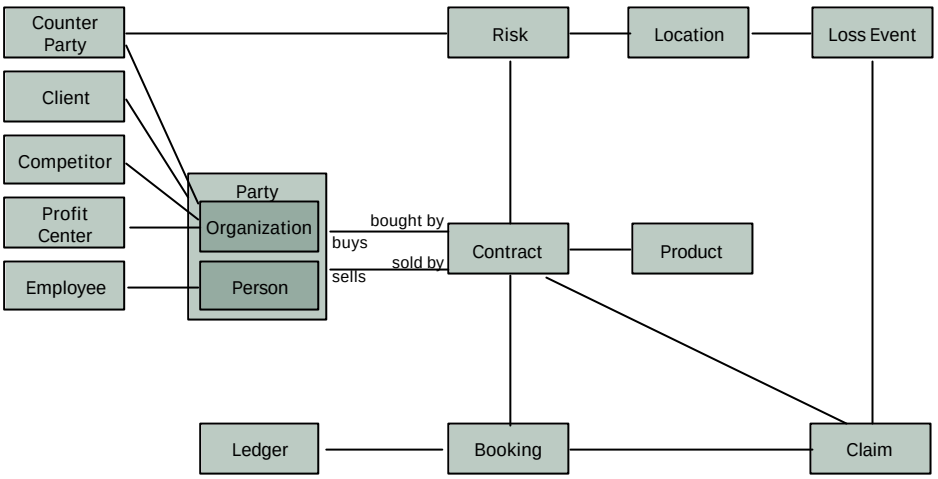


Figure 4: High-Level Data Model for Reinsurance

Note that relationships in this diagram are generally considered to be many-to-many relationships, with the exception of the relationships between entity type Party and its subtypes on one hand and the entity types which establish the roles that can be played by a Party on the other. This data model has served as a foundation to design the physical data model of both a Data Warehouse project as well as a new project to establish an Operational Data Store (ODS) which mainly serves as a data exchange hub.

6 Stable Elements of a Data Model

The entity types in a data model can be classified according to their usage as follows (see also Figure 5):

- *Interaction entity types* are entity types which describe business interactions between the enterprise and an outside entity such as a client, a supplier or some sort of intermediary. Examples of interaction entity types are Quote, Contract, Booking. (Note that interaction entity types contain information which is typically stored in the fact tables of a data mart [KR02]. Most importantly, they comprise key business performance indicators, e.g., in reinsurance, premium_earned, losses_incurred, and cost.)
- *Context entity types* are entity types which describe the context of business interactions. Examples of context entity types are Product, Client, Employee, ProfitCenter. (Context entity types contain data which is typically stored in dimension tables of a data mart [KR02].)

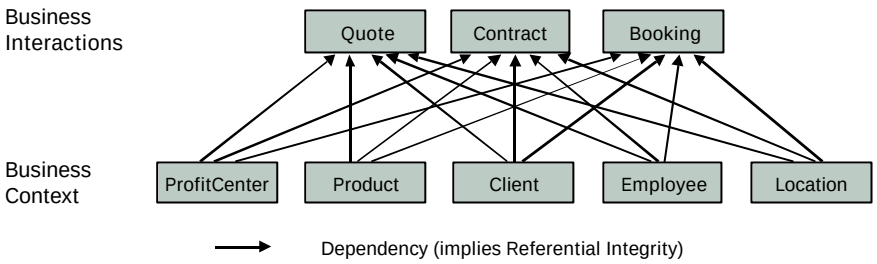


Figure 5: Interaction Entity Types depend on Context Entity Types

The stable elements of an enterprise information architecture are to a large extent determined by the structure and (predefined) content of context entity types:

- *Identifiers of context entity types*
  Global identification schemes of context entity types serve as the glue to tie together related information scattered across data stores controlled by the various applications. Examples are enterprise-wide IDs for clients, employees etc. These identifiers serve as surrogates for real-world entities and as such (1) must be devoid of any semantics, (2) must not change during the lifetime of the corresponding real-world entity, and (3) must not be reused for another entity.
  Note, however, that establishing a global identification scheme alone is not good enough. For example, even with a single globally used client management system such as Swiss Re's CMS, it is still possible, depending on corporate culture and established processes, that one client is represented multiple times in the database.

- *Domains of attributes of context entity types*
  In order to segment the values of key business performance indicators according to the properties of context entity types, a common terminology has to be established. An important part of this terminology is defined by the *domains* of attributes which describe context entity types (see also Figure 6). Typically, these domains look like enumeration types in programming languages such as C or Pascal in that they consist of a predefined set of data values.
  In addition, within a domain, there are often taxonomic relationships between data values in that a data value representing a more general concept is related to a number of data values representing more specific concepts. While data is usually entered into OLTP systems using the most specific concepts of a domain only, terms describing more general concepts are typically used for analysis and reporting purposes.
  Example (see also Figure 7): The domain LineOfBusiness consists of a predefined set of terms (enumeration constants) such as 'property', 'casualty', 'engineering', 'marine', 'life', 'health' etc. However, smaller lines such as 'engineering' and 'marine' may be subsumed by the more general term 'special line'.

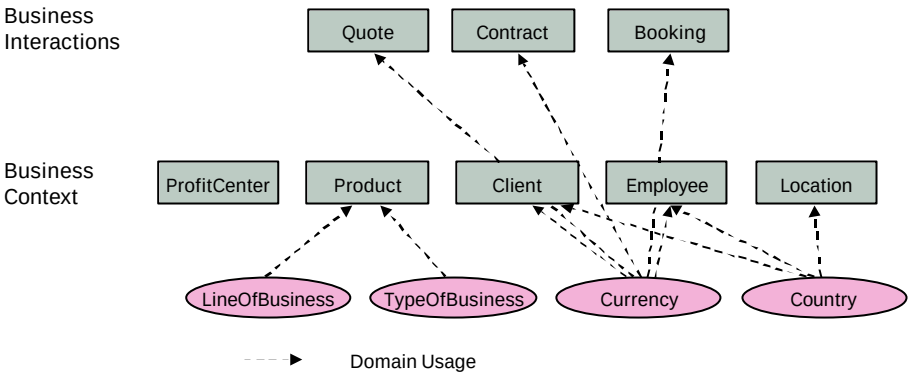


Figure 6: Domains and their Usage by Interaction and Context Entity Types

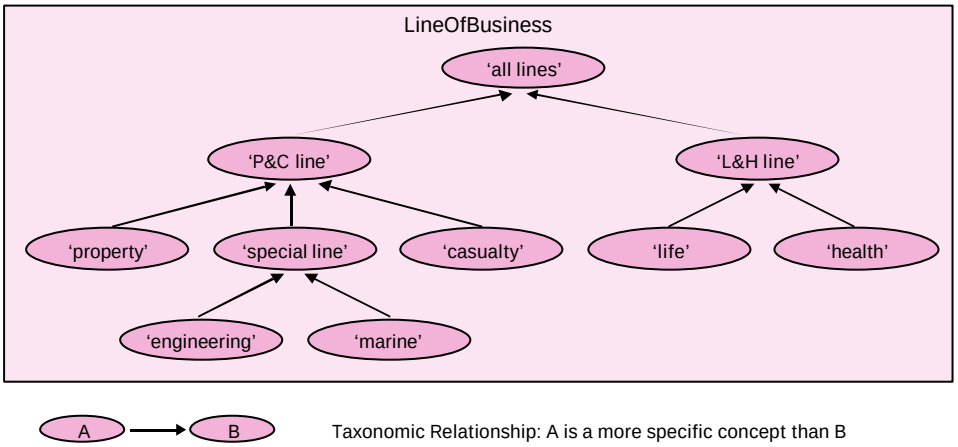


Figure 7: Taxonomic Relationships between Data Values of a Domain

7 Repository for Metadata and Reference Data

Regarding the integration of enterprise applications, there have been many proposals for technologies to solve interoperability issues, e.g., CORBA, J2EE, Web Services, etc. While these technologies are great enablers, they cannot directly solve the much more important and difficult issues to achieve interoperability on a semantic level. (This is even true of XML, which is being touted as the solution to all interoperability problems, beacuse the real issue is to agree on the semantics of the XML tags used to describe information which is being exchanged.)

Since migrating existing operational applications from current local identification schemes and terms to their global counterparts is often non-trivial, mappings from local

identifiers and terms to the corresponding global identifiers and terms must be supported.

Moreover, a *central repository for metadata and reference data* should contain the definitions of terms used to describe data structures and (predefined) content, their usage in applications, and mappings between these terms.

As an aside, note that it is sometimes difficult to clearly distinguish metadata and reference data: Typically, metadata is defined as "data that describes or specifies other data". As an example, the names of tables and columns in a relational database setting are considered metadata. Reference data is defined in [Ch01] as "any kind of data that is used solely to categorize other data found in a database, or solely for relating data in a database to information beyond the boundaries of the enterprise", i.e., consists of domains and predefined data values as discussed in the previous section. However, depending on the design of a database, what is metadata in one design can be reference data in the other, as illustrated by the terms on gray background in the following figure which depicts two alternative database designs:

GrossNetV1

Treaty Year	Business Year	LoB	ToB	Premiums	Claims
1999	1999	'Property'	'proportional'	1654	-1623
...	...	...	...	...	...

GrossNetV2

Treaty Year	Business Year	LoB	ToB	Key Figure Name	Key Figure Value
1999	1999	'Property'	'proportional'	'Premiums'	1654
1999	1999	'Property'	'proportional'	'Claims'	-1623
...	...	...	...	...	...

Figure 8: The same business terms may appear as metadata or as reference data

The ultimate goal is to administer all metadata and reference data used in various systems in the central repository so that it serves as a single point of reference for all applications. This central repository should also keep track of the *history of reference data* in order to support the analysis of key business performance indicators over time, notwithstanding changes to the attributes of context entities, e.g., as induced by the change of location of a client, or an organizational realignment.

The current tool for the management of reference data at Swiss Re, SDL Tool Release 3, only supports (1) the definition of business terms, (2) browsing and querying available business terms over the intranet, and (3) the generation of a manual which specifies the structure for the collection of financial data. With one exception, reference data is currently still transferred to operational systems in a manual process. Also, there is no sup-

port for historization yet. The system is built on top of relational technology (DB2 on OS/390), given that with a few exceptions, almost all key applications within Swiss Re are also built on relational technology (DB2 on OS/390 or Oracle on Solaris).

The design and implementation of a new version of SDL Tool (R4) has been started. It will contain various improvements on a technical level, most importantly regarding the release process of consistent sets of reference data, including various APIs to allow other applications to retrieve reference data from the central repository, and support for versioning and historization of metadata. An excerpt of the SDL R4 data model is shown in Figure 9.

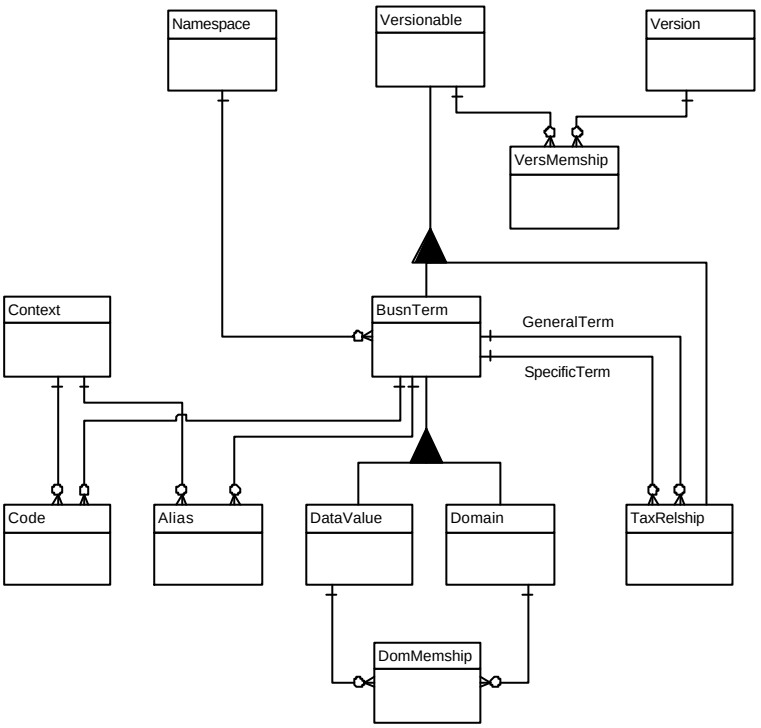


Figure 9: Schema of SDL Tool R4 (Excerpt)

With respect to the historization of reference data, we'd like to mention in passing that having a temporal query languages such as TSQL2 or SQL/Temporal [Sn00] is not nearly as important as establishing a good design of a data model which supports historization. The basic idea is to represent the lifespan of an entity together with its "unchanging" attributes in one table, and each group of attributes which changes simultaneously in a separate table (see e.g. [Ma94], or, more accessibly, [Sn00] Chapter 11 and [DD03] Chapter 10 for a discussion of design of temporal databases).

8 Conclusions

In summary, information integration in a global enterprise is based on (1) a high-level data model, (2) global identification schemes of relevant (context) entity types, and (3) a global terminology for the relevant domains of important attributes, where relevance is to a large extent established by global business needs. Of course, even such a minimalist approach to information integration is not without challenges. As [MK01] eloquently point out, success of IT projects and systems in corporations depends not only on having the right technology, but also on having the right content, and, most importantly, an information culture which encourages information sharing across organizational boundaries.

Acknowledgements

The author would like to thank Benno Büeler and Hans Wegener for their work on SDL Tool R4, and P. Cartwright, T. Harris, M. Kaufmann, A. Mönkeberg for their contributions to the topic of metadata and reference data management in general and earlier releases of SDL Tool in particular.

References

[Ch01] M. Chisholm: *Managing Reference Data in Enterprise Databases*. Morgan Kaufmann, 2001.

[CL00] R. L. Carter, L. D. Lucas, N. Ralph: *Reinsurance, 4th Edition*. Reactions Publishing Group / Guy Carpenter & Co., 2000.

[Co96] M. A. Cook: *Building Enterprise Information Architectures: Reengineering Information Systems*. Prentice Hall, 1996.

[DD03] C. J. Date, H. Darwen, N. A. Lorentzos: *Temporal Data and the Relational Model*. Morgan Kaufmann, 2003.

[Fo97] M. Fowler: *Analysis Patterns – Reusable Object Models*. Addison Wesley, 1997.

[GV96] GDV (Gesamtverband der Deutschen Versicherungswirtschaft e.V.): *VAA – Die Anwendungsarchitektur der Versicherungswirtschaft*. VDS Dienstleistungs-GmbH, 1996.

[IB02] IBM Corp.: *IBM Insurance Application Architecture (IAA)*. http://www-1.ibm.com/industries/financialservices/doc/content/solution/278918103.html

[Ha96] D.C. Hay: *Data Model Patterns – Conventions of Thought*. Dorset House, 1996.

[Ha03] D. C. Hay: *Requirements Analysis: From Business Views to Architecture*. Prentice Hall, 2003.

[KR02] R. Kimball, M. Ross: *The Data Warehouse Toolkit, 2nd Edition.* John Wiley & Sons, 2002.

[Ma94] R. Marti: Entwurf Temporaler Datenbanken. *Fortbildungskurs Temporale Datenbanken*, ETH Zürich, 1994.

[MK01] D. A. Marchand, W. J. Kettinger, J. D. Rollins: *Making the Invisible Visible: How Companies Win with the Right Information, People and IT*. John Wiley & Sons, 2001.

[Si01] L. Silverston: *The Data Model Resource Book* (Vols. 1 and 2). John Wiley & Sons, 2001.

[SH92] S. H. Spewak, S. C. Hill: *Enterprise Architecture Planning: Developing a Blueprint for Data, Applications and Technology*. John Wiley & Sons, 1992.

[Sn00] R. T. Snodgrass: *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, 2000.

[Za87] J. A. Zachman: A Framework for Information Systems Architecture. *IBM Systems Journal*, Vol. 26, No. 3, 1987.

Integration von ETL und OLAP in die relationale DWH-Technologie: mehr Lösung für weniger Aufwand?

Ein Datawarehouse - Praxispapier

Marc Bastien

Oracle Deutschland GmbH
Notkestrasse 15
D-22607 Hamburg
Marc.Bastien@oracle.com

Abstract: Seit vielen Jahren werden Lösungen für Datawarehouse Themen schon nach dem gleichen Schema entwickelt. Dies ist nicht weiter verwunderlich, da doch die Anforderungen –eigentlich- immer ähnlich sind: Daten aus verschiedenen Quellsystemen sollen, unter diversen Randbedingungen, analysiert werden. Auch wenn sich Technologien weiter entwickelt haben, Schnittstellen nun bereits out-of-the-box verfügbar sind und diverse grafische Oberflächen die Entwicklung und den Betrieb vereinfachen, so ist der Weg der Daten in das Warehouse bis zum Nutzer weitgehend gleich geblieben: Extrakt der Daten aus den diversen Quellen, Laden und Überführen der Daten in ein relationales Datenbanksystem, dort die Speicherung in Tabellen, eventuell Überführung aller, oder einiger Daten in eine multidimensionale Datenbank, die Speicherung in „Cubes“, die Berechnung von Kennzahlen und schließlich die Ausgabe der Daten mehr oder weniger übersichtlich an den Benutzer. Dieser Artikel beschreibt am Beispiel eines Controlling Datawarehouse, wie man durch die Integration der verschiedenen Technologien (ETL, Relational und OLAP) in einer Oracle Datenbank, Release 9*i,* bei gleichem Funktionsumfang wesentlich bei der Realisierung und Betrieb vereinfachen (=sparen) kann.

1 Einleitung

Datawarehouse (DWH) kommt nicht aus der Mode. Wurde es vor zehn Jahren noch als Erfolg gefeiert, aus einem, oder vielleicht zwei operativen Systemen Daten innerhalb einer Woche zu extrahieren, zu transformieren, in relationale Tabellen möglichst platzsparend zu laden und einem sehr ausgewählten Benutzerkreis per Text-orientierter Abfragesprache zur Verfügung zu stellen, so sind es heute andere Anforderungen, die ein DWH zu einem fachlich und technisch anspruchsvollen Projekt machen.

1.1 Die Schere wird breiter: Anforderungen an ein modernes Datawarehouse

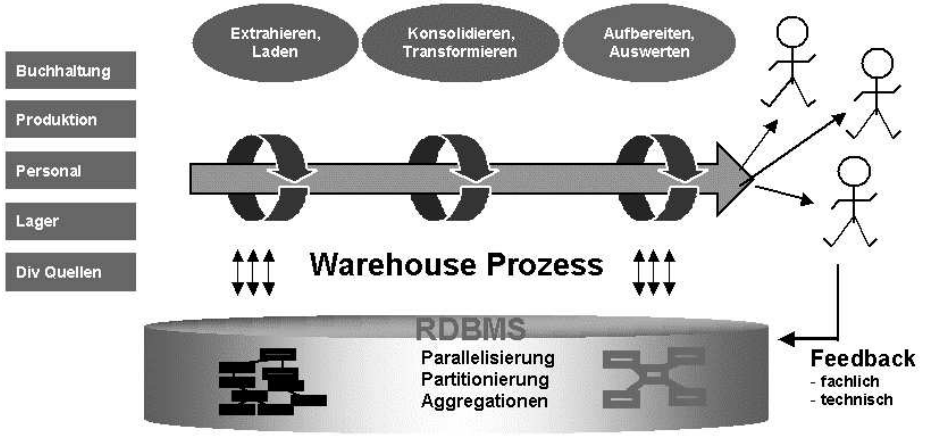
Ein modernes DWH muss vielen Anforderungen genügen. Dazu gehören natürlich alle bekannten Anforderungen, die auch in der Vergangenheit schon wichtig waren. Ich möchte diese unter dem Begriff „*Daten so bereitstellen, dass sie verstanden und analysiert werden können*“ zusammenfassen.

Dazu kommen neuen Anforderungen, die ein DWH-Projekt anspruchsvoll machen:
- Mehr Daten: DWH im zwei bis dreistelligen Terabytebereich stellen bald keine Seltenheit mehr da, häufig wird mit wenig Daten begonnen und dann gesteigert
- Mehr Benutzer: vierstellige Benutzerzahlen muss ein modernes DWH verkraften können, ohne dass die Antwortzeiten sich merklich verlängern.
- Höhere Flexibilität: neue Datenquellen sollen kürzester Zeit, zusammen mit den bisherigen Daten, den Benutzern vorliegen.
- Operationalisierung: (wie es OLTP-Systeme vormachen) in minimaler Zeit die Daten den DWH-Benutzern verfügbar machen (Stichwort: Real-Time Datawarehouse)

Aber es gibt auch gegensätzliche Trends, die einfachere DWH-Projekte erfordern
- Strengere Betrachtungen des ROI: lohnt sich ein DWH? Wenn man den Nutzen schlecht beziffern kann, wie sollen die Kosten beurteilt werden?
- Enge Personalressourcen: durch Personalreduktion in IT-Abteilungen sind weniger IT - Projektmitarbeiter verfügbar; Für ein DWH stehen nun oft die Fachanwender bereit, die einen Teil der Zeit, in der sie z.B. mit ihren Kunden im Kontakt sein sollten, für die Arbeit am DWH aufwenden.
- Kürzere Entwicklungszeiten: lange Projekte kosten Ressourcen und die Gefahr, den Projektfokus aus den Augen zu verlieren, steigt mit der Projektlaufzeit.

1.2 Der Datawarehouse Prozess

Aus den o.g. Anforderungen erwächst der Bedarf, die Vorgehensweise bei der Erstellung eines DWH zu überdenken: ist es immer sinnvoll und notwendig, alle Bereiche (ETL, Datenbank, OLAP, Endbenutzerwerkzeuge, Business Intelligence - Tools) einzeln zu betrachten? Oder kann es vielmehr sinnvoll sein, DWH als Prozess zu verstehen:



Durch diesen Ansatz wäre man nicht ständig bestrebt, für jeden der Teilbereiche die beste, zu diesem Zeitpunkt gerade verfügbare, Technologie zu verwenden, sondern könnte sich auf das Wesentliche konzentrieren, nämlich die Erstellung einer Lösung für die Anwender, mit weniger Aufwand bei der Erstellung und Wartung.
Für die Technologie, die diesen Ansatz unterstützen soll, heißt dies, dass sie mit Hilfe von Metadatenrepositories und verbundenen Schnittstellen (am besten integriert) agieren muss: Metadaten müssen durch ein Werkzeug angelegt werden, und allen anderen Komponenten verfügbar gemacht werden. Wenn erst einmal die verschiedenen Bereiche („Stage“, „Warehouse“, „Mart“) und deren Abhängigkeiten (Verbindungen) innerhalb des DWH durch Dimensionen und Kennzahlen aber auch durch Tabellen, Views, Prozeduren usw. beschrieben sind, können die alle weiteren Prozesse innerhalb des DWH leicht abgeleitet und generiert werden. Erweiterungen oder Änderungen werden dann im Repository vorgenommen, generiert und kurze Zeit später dem Endbenutzer zur Verfügung gestellt. Ebenso wird die Portierung auf andere Umgebungen erleichtert: die Metadaten beschreiben das DWH plattform-unabhängig und ermöglichen quasi ein portables Warehouse.

2 Der Praxisfall

2.1 Beschreibung des Problems

In einem internationalen Großkonzern im Bereich Maschinenbau wurde das unternehmensweite Controlling ausschließlich auf der Basis von Microsoft Excel bzw. integrierte Plug-Ins zu SAP R/3 durchgeführt. Kennzahlen wurden durch die Controller in den Berichten in MS Excel als Makros implementiert. Die erzeugten Berichte wiesen z.T. Fehler auf, waren zu spät und nur einem eingeschränkten Benutzerkreis zugänglich. Sollten neue Berichte erzeugt werden, mussten sich mehrere Controller tagelang nur mit der Erstellung der Berichte und der entsprechenden Formatierung beschäftigen. Die gedruckten Berichte wurden per Post innerhalb und außerhalb Deutschlands verteilt. Es liegt auf der Hand, dass der deutsche Vorstand seine Berichte eher verfügbar hatte, als sein südafrikanischer Kollege.

2.2 Das Projekt

Im Rahmen eines konzernweiten Projektes wurde die Einführung eines unternehmensweiten Controlling DWH beschlossen. Wesentliche Eigenschaften sollten sein:

- geprüfte Routinen für das Einladen der Daten aus SAP R/3 in das DWH, kein manueller Eingriff, lückenlose Verfolgung der Routinen
- DWH-gerechtes Datenbank Schema, ausbaubar, pflegbar, nutzbar
- Abdeckung neue Anforderungen an das Reporting:
  - Neue, umfangreiche Kennzahlensysteme (GuV, Erfolgsrechnungen)
  - Möglichkeit zur Prognose
  - Erweiterte Analysemöglichkeiten
  - Speicherbare Auswertungen
  - Zeitreihenbetrachtung mit verschiedenen Szenarien
- Konzernübergreifendes Repository für Kennzahlen, Benutzer, Benutzerrechte und Standardberichte
- Zugriff über das Intranet auf
  - Standardberichte: druckbar, mit einheitlichem Layout, anhand von Parametern veränderbar, auf Knopfdruck verfügbar (Zielgruppe: bis zum Top-Management)
  - Ad-Hoc Berichte: Mehrdimensionale Selektion der Daten und Berichte, auch dort u.a. vordefinierte Berichte, die aber individuell gestaltbar sein sollen (Zielgruppe: Controller und Analysten aus allen berechtigten Bereichen)

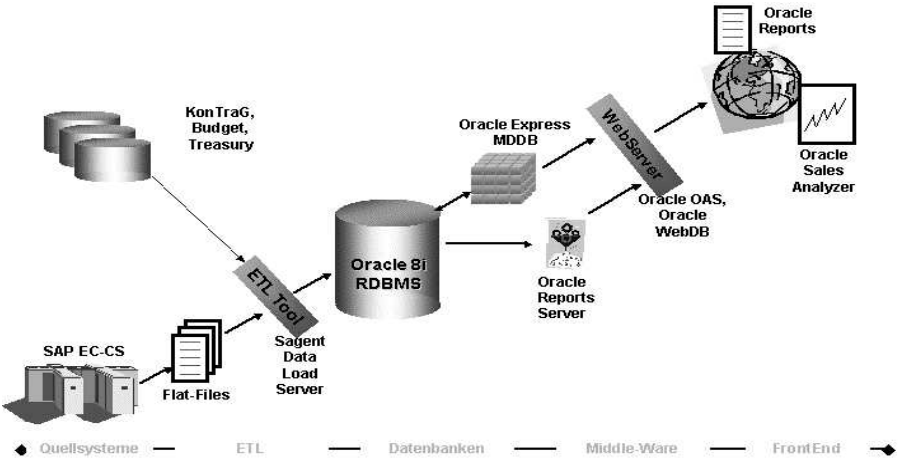
2.3 Die realisierte Lösung

2.3.1 Beschreibung

Es wurde ein zentrales, Intranet basierendes Management Information Portal für konsistentes Reporting, Benutzergruppen-abhängigen Inhalt inklusive neuer Kennzahlen und Berechnungen für Management und Mitarbeiter des Controlling (ca. 500 Benutzer) erstellt. Für die Auswahl der Werkzeuge wurde ein Proof-of-Concept (PoC) mit namhaften Anbietern von Datawarehouse Lösungen durchgeführt. Schon im Vorfeld wurde sich gegen eine Implementierung von SAP BW entschieden, da nicht alle Anforderungen erfüllt wurden.
Als Sieger aus dem PoC gingen folgenden Hersteller mit ihren Werkzeugen hervor:

- ETL Prozess: Sagent Data Load Server 4.2
- Datenspeicherung (relational): Oracle RDBMS 8.1.6
- Datenspeicherung (multidimensional): Oracle Express 6.3
- Middleware, WebServer: Oracle Application Server 4.0.8.1
- Auswertung, Portal: Oracle WebDB 2.2
- Auswertung, Standardberichte: Oracle Reports 6i
- Auswertung, Ad-Hoc: Oracle Sales Analyzer 11i

2.3.2 Lösungsarchitektur (Hardware/Betriebssystem): Sun / Solaris 4.6



2.3.3 Datenfluss

- die Daten werden durch SAP EC-CS als ASCII-Dateien bereit gestellt
- die Dateien werden durch das ETL Werkzeug Sagent erkannt und in die „Staging-Area“ der Oracle Datenbank geladen
- die Daten werden bereinigt, aufbereitet und in das DWH Schema überführt
- Daten werden z.T. in die mehrdimensionale Datenbank Oracle Express kopiert
- In Express wird die Verdichtung über alle Hierarchien (Aggregation), sowie die Berechnung aller Kennzahlen durchgeführt
- Die verdichteten Daten und die berechneten Kennzahlen werden aus Express zurück in die relationale Datenbank exportiert
- Zugriff auf die mehrdimensionalen Daten durch Oracle Sales Analyzer (Ad-Hoc Reporting) und Zugriff auf die relationalen Daten durch Oracle Reports (Standardberichte)

2.3.4 Vorteile dieser Lösung

Durch den Einsatz der Oracle RDBMS war zu keinem Zeitpunkt in Frage gestellt, ob das aufkommende Datenvolumen aufgenommen werden konnte, oder nicht. Durch die zusätzliche mehrdimensionale Datenhaltung ergänzte man die Vorteile der relationalen Datenhaltung um die Möglichkeit, Daten auch multidimensional auszuwerten. Die multidimensionale Datenbank Express lieferte durch die Applikation Oracle Sales Analyzer (OSA) eine so flexible Analyse, wie sie mit rein relationalen Werkzeugen nicht möglich war. Durch die Nutzung von OSA war die Speicherung der Daten in einer MDDB vorgegeben, deshalb konnten die bereits in Oracle Express enthaltenen Prozeduren zur Aggregation über mehrdimensionale Hierarchien und Kennzahlenberechnung genutzt werden und man brauchte keinen Aufwand in die komplizierte Entwicklung der Kennzahlenberechnung und Aggregation in der RDBMS stecken.
Durch die bereits reichlich vorhandenen analytischen Funktionen und Programmen konnte auf umfangreiche Entwicklung in Oracle Express verzichtet werden, nur für die Steuerung der Berechnungen und für den Import und Export mussten Programme entwickelt werden. Dabei erwies es sich als Vorteil, dass direkt in der MDDB Programme erstellt werden konnten, die in der Endausbaustufe über UNIX-Scripts direkt aus Sagent aufgerufen werden konnten. So war durchgängig abgesichert, dass alle Prozesse in der richtigen Reihenfolge ablaufen und, wenn nicht, eindeutig den Stand der Verarbeitung bestimmt werden konnte. Die Geschwindigkeit der Kennzahlenberechnung und der Aggregation war trotz der hohen Datenmengen äußerst zufriedenstellend.

2.3.5 Nachteile dieser Lösung

Architektur: es waren zwar, wie damals häufig üblich, für alle Teilbereiche optimale Lösungsbausteine gewählt, an deren individuellen Leistungsfähigkeit auch nichts auszusetzen war, aber durch die zahlreichen, nicht genormten Schnittstellen gingen gewonnene Vorteile verloren, und die Komplexität des Projektes stieg deutlich. Das Projekt dauerte länger und erforderte mehr Skills.
Metadaten: Es gab in dem ETL Werkzeug nicht die Möglichkeit, standardisierte Warehouse-Metadaten abzulegen. Dies führte dazu, dass in der relationalen Struktur nicht erkennbar war, welche Dimensionen oder Fakten vorhanden waren. Alle Informationen dazu waren nur in den Köpfen der Entwickler bzw. in der Dokumentation nachzulesen. Die Projektlaufzeit verlängerte sich, da einige Schnittstellen häufig umgeschrieben werden mussten, weil sich Entwickler missverstanden.
Datensicherung: Durch die völlig unterschiedliche Technologien der RDBMS und MDDB mussten Backup und Recovery auch hier komplett getrennt betrachtet werden. Abweichungen führten z.T. dazu, dass die MDDB nicht zurückgesichert werden konnte, sondern komplett neu aus der RDBMS befüllt werden musste.
Sicherheit: durch die unterschiedlichen Sicherheitskonzepte von RDBMS und MDDB mussten Programme entwickelt werden, die die Benutzerrechte aus einem zentralen Repository lesen und einsetzen konnten.

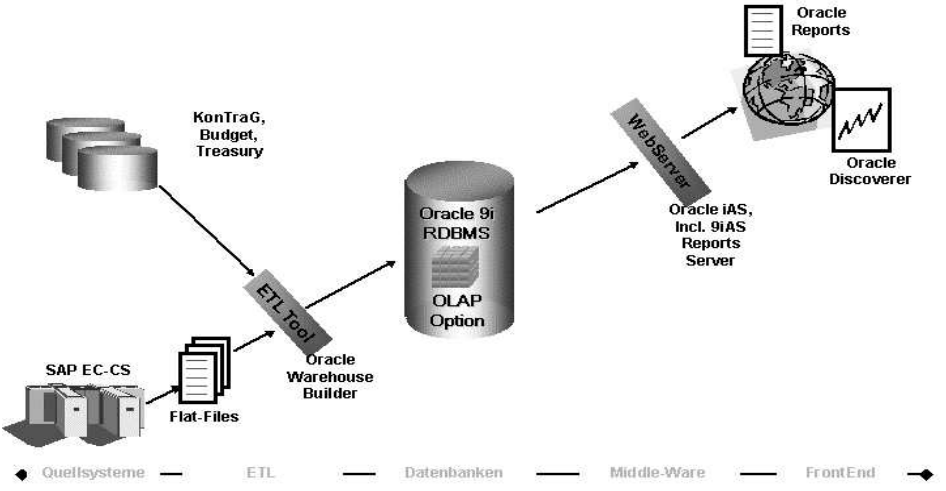
2.4 Eine alternative Lösung

2.4.1 Beschreibung

Für die alternative Lösung ist zu beachten, dass, während die Nachteile reduziert, oder eliminiert werden, die Vorteile der o.g. Lösung erhalten bleiben. Durch die Weiterentwicklung der DWH-Funktionalitäten in der Oracle 9*i* Datenbank und die Entwicklung von passenden Werkzeugen ist es möglich, einen komplexen Warehouse Prozess zu implementieren, und nicht Einzelkomponenten miteinander zu verbinden.
Die Lösungskomponenten der neuen Lösung:

- ETL Prozess: Oracle Warehouse Builder 9i
- Datenspeicherung (relational + OLAP): Oracle RDBMS 9i + OLAP Option
- Middleware, WebServer: Oracle 9i Application Server 9i
- Auswertung, Standardberichte: Oracle Reports 9i
- Auswertung, Ad-Hoc: Oracle Discoverer 9i

2.4.2 Lösungsarchitektur der Alternative



2.4.3 Datenfluss

- die Daten werden durch SAP EC-CS als ASCII-Dateien bereit gestellt
- die Dateien werden durch das ETL Werkzeug Oracle Warehousebuilder erkannt und in direkt in das DWH-Schema geladen. Während dieses Prozesses werden die Daten bereinigt und aufbereitet
- Teile der Daten werden in den „Analytic Workspace“ (den mehrdimensionalen Bereich) der relationalen Datenbank überführt
- Im Analytic Workspace wird die Verdichtung über alle Hierarchien, sowie die Berechnung alle Kennzahlen durchgeführt
- Auf die Daten in den relationalen Tabellen und im Analytic Workspace kann direkt mit allen SQL-Werkzeugen zugegriffen werden

2.4.4 Vorteile der neuen Lösung

Die grundsätzliche Architektur bleibt unverändert, weshalb es auch nicht verwundert, dass die Vorteile sich nicht verändern: Datenspeicherung relational und mehrdimensionale Kennzahlenberechnungen.
Zusätzlich ergeben sich aber neue Vorteile:
Metadaten: der Warehousebuilder erzeugt beim Anlegen des Schemas und der Verbindungen zwischen Quelle und Ziel bereits alle Metadaten für das DWH. Die Metadaten sind durchgängig und vom ETL-Prozeß bis zur Auswertung nutzbar. Für die wenige Ausnahmen sind fertige „Bridges" definiert, die die zusätzlichen Daten automatisch erzeugen. Die durchgängigen Metadaten ermöglichen zusätzlich eine Herkunftsanalyse, d.h. es kann jederzeit ermittelt werden, aus welcher Quelle welche Daten stammen.
eine integrierte Technologie: alle Daten befinden sich in einer Datenbank, d.h. die Metadaten, die Tabellen mit den Warehouse Daten und selbst die mehrdimensionalen Daten sind in einer DB gespeichert. Dies bedeutet nicht nur Vorteile bei der Wartbarkeit oder der Performanz, sondern auch, dass jetzt nur ein Backup/Recovery Konzept entwickelt und umgesetzt werden muss: wird die Datenbank gesichert, werden alle Daten, egal, ob relationale Tabellen, mehrdimensionale Objekte oder Metadaten gesichert. Des weiteren bietet die integrierte Technologie natürlich auch ein integriertes Benutzerkonzept, d.h. es gibt keine Notwendigkeit für eine doppelte Benutzerverwaltung in RDBMS und MDDB.
Integriertes OLAP bedeutet auch, dass die Daten des OLAP Würfels nun allen Benutzern zur Verfügung stehen. Es werden keine speziellen Applikationen benötigt, da alle Daten per SQL im Zugriff sind. Vorher benötigte Schnittstellen von der MDDB zur RDBMS können entfallen.

2.4.5 Mögliche Nachteile der neuen Lösung

Migration: der Aufwand für die Migration ist im Abschnitt „Aufwand für die Migration".
Frontend: Oracle Sales Analyzer (OSA) basiert auf der Express Technologie und kann nicht zusammen mit 9i OLAP eingesetzt werden. Es muss auf eine andere Applikation, den Oracle Discoverer, zurückgegriffen werden, was ich hier unterstellt habe. Dies würde eine Umstellung der Endanwender erfordern.
Alternativ könnte eine eigene Applikation erstellt werden, die OSA gleicht. Dazu böte sich der Oracle JDeveloper an, der, zusammen mit den Oracle BI-Beans, mit fast gleichen Analysemöglichkeiten (verglichen zu OSA) aufwarten kann.
Beide Werkzeuge setzen natürlich auf den Metadaten vom Warehousebuilder auf.

2.4.6 Aufwand für eine Migration

Der Aufwand einer Migration kann in mehrere Teile zerlegt werden:

- ETL: der komplette ETL Prozess muss neu im Oracle Warehousebuilder erstellt werden. Falls die vorher verwendete Lösung standardisierte Metadaten (CWM) unterstützen würde, wäre ein Import möglich.
- Datenbank: kaum Aufwand, da nur ein Upgrade von 8i nach 9i erfolgen muss
- OLAP: alle Programme und Daten können nach 9i OLAP übernommen werden, allerdings müssen Anpassungen auf die neuen Metadaten vorgenommen werden
- Middleware: ein Upgrade von Oracle OAS nach iAS ist möglich
- Frontend
    - Oracle Reports: Upgrade nach 9i ist mit wenig Aufwand möglich
    - Oracle Sales Analyzer nach Discoverer: alle Berichte müssen neu erstellt werden
    - (alternativ) OSA nach eigene Applikation: eine komplette Applikation kann mit dem JDeveloper und BI-Beans neu entwickelt werden

3 Fazit

Für den konkreten Fall ist sicherlich entscheidend, wie hoch der tatsächliche Aufwand für die Migration wäre, schließlich sind bereits alle Applikationen und Schnittstellen implementiert. Wahrscheinlich bietet es sich an, die Migration (oder Teile hiervon) im Rahmen von sowieso geplanten Änderungen oder Upgrades an der Lösung durchzuführen.

Für alle neuen Projekte zeigt dieses Beispiel wie viel leichter, und damit billiger, sich DWH – Projekte sich mit den Komponenten:

1) integrierte Metadaten
2) integrierte Datenbanktechnologie
3) angepasste Tools für ETL und BI

durchführen lassen können. Schließlich hat sich gezeigt, dass nicht Hard- und Software den Preis eines Projektes maßgeblich beeinflussen, sondern Dauer und Komplexität der Implementierung und des Betriebes.

Skalierbare Verarbeitung von XML mit Infonyte-DB[1]

Thomas Tesch, Peter Fankhauser, Tim Weitzel

Infonyte GmbH
Julius-Reiber-Str. 15
D-64293 Darmstadt
Tel. +49 (0) 700 INFONYTE
info@infonyte.de

Zusammenfassung: Die zunehmende Durchdringung von IT-Architekturen mit XML führt zu immer größeren XML-Datenvolumen. Diese lassen sich mit den zur Verfügung stehenden XML-Werkzeugen nicht ausreichend skalierbar verarbeiten. Dieser Beitrag stellt den persistenten XML Prozessor Infonyte-DB vor, der auch große XML-Datenvolumen resourcenschonend und effizient verarbeiten kann, und illustriert an Hand von konkreten Anwendungsszenarien seinen Einsatz für XML-basierte Webservices, technische Dokumentation, sowie mobiles Informations-management.

Abstract: The emerging penetration of IT architectures with XML leads to increasing XML data volumes. Available tools often fail in realizing scalable XML processing for large XML data volumes. This paper introduces Infonyte-DB, a persistent XML Processor that economizes on system resources and allows processing large XML data volumes. Based on concrete application scenarios it illustrates, how Infonyte-DB can be deployed for XML based web services, technical documentation, and mobile information management.

1 Einleitung

Die Auszeichnungssprache XML hat sich in den letzten Jahren nicht nur in ihrer angestammten Domäne des Dokumentenmanagements sondern auch zum Nachrichtenaustausch im E-Business-Bereich [RaAS02], zur Konvertierung heterogener Datenbestände [BuLW01] sowie allgemein als plattform- und programmiersprachenunabhängige Basistechnologie zum Datenaustausch wie beispielsweise in Web-Service-Architekturen [BeMW02] durchgesetzt.

Dieser Beitrag illustriert die Rolle von XML bei der Kopplung von IT-Systemen und verdeutlicht die Grenzen von existierenden XML-Werkzeugen bei der Verarbeitung von großen Datenmengen. So stoßen etwa gängige Prozessoren zur Transformation von XML-Daten bei wachsendem Datenvolumen rasch an Leistungsgrenzen. Es stellt sich die Frage nach skalierbaren Werkzeugen zur Speicherung und Verarbeitung von XML-Daten.

[1] Eine frühere Version dieses Beitrags erschien in der Zeitschrift Wirtschaftsinformatik WI 5/2002

Vor diesem Hintegrund wird die native XML-Datenbank der Infonyte GmbH als persistenter XML-Prozessor vorgestellt, der insbesondere die effiziente Verarbeitung großer XML-Datenvolumen ermöglicht und dabei sehr geringe Systemanforderungen stellt. Zentrale technische Grundlagen sind das am Fraunhofer Institut für Integrierte Publikations- und Informationssysteme (IPSI) entwickelte persistente DOM (PDOM, eine persistente Implementierung der W3C-DOM-Schnittstelle) sowie ein webfähiger Anfrageprozessor, der alle gängigen XML-Anfrage- und -Transformationssprachen unterstützt. Die Infonyte GmbH mit Sitz in Darmstadt ist ein Spin-Off des Fraunhofer IPSI.

Die Infonyte-Technologie wird in einer Vielzahl von Produkten wie Web-Portalen, B2B-Nachrichtensystemen, Content Management Systemen und mobilen Informationssystemen erfolgreich eingesetzt. Neben vor allem amerikanischen Luftfahrtunternehmen verwendet eine wachsende Zahl von Kunden aus der IT-Industrie, der Fertigungsindustrie und dem Finanzbereich Infonyte-DB als Komponente in ihren Produkten.

2 Herausforderungen bei der XML-Verarbeitung

Die Herausforderung heutiger IT-Systemarchitekturen ist die kostengünstige und flexible Kopplung von autonomen und heterogenen Systemen. Dabei sind in der Hauptsache Medien- und Prozessbrüche zu überwinden. XML hat sich hier als neutrales Datenaustauschformat durchgesetzt. Mit XML lassen sich auch komplexe Datenstrukturen vergleichsweise einfach darstellen, mit ihren Metadaten kombinieren, und zwischen verschiedenen Plattformen und Programmiersprachen austauschen. Für den praktischen Einsatz gibt es eine breite Basis von freien und kommerziellen Werkzeugen zur Erstellung, Validierung und Transformation von XML. In zwei Bereichen kommen die Stärken von XML besonders zum Tragen:

Medienneutrales Publizieren: Mit der zugrundeliegenden Trennung von Inhalt und Darstellungsinformationen hat XML seine Kernanwendungsdomäne im Bereich von komplexen medienneutralen Publikationsprozessen. Sowohl für Content-Management-Systeme, die Informationen bedarfsgerecht für bestimmte Zielmedien zusammensetzen, als auch Dokumentenmanagementsysteme, die ihren Schwerpunkt in der Verwaltung, Archivierung und Abfrage großer Dokumentenbestände haben, ist XML die ideale Basistechnologie. Dabei müssen häufig verschiedene proprietäre Dokumentenformate vor der Weiterverarbeitung in eine XML-Repräsentation gebracht werden. Mit entsprechenden Transformationssprachen wie der Extensible Stylesheet Language Transformations (XSLT) lassen sich aus den XML-Dokumenten dann beliebige Präsentationsformate (HTML, PDF, eBook-Formate, Druckvorstufe) erzeugen.

Kopplung von Geschäftsprozessen: Bei der innerbetrieblichen Kopplung von Geschäftsprozessen unter dem Schlagwort Enterprise Application Integration (EAI) werden Lösungen zur Überwindung von Unterschieden zwischen Mainframe-Systemen, Legacy-Anwendungen, ERP-Software und Web-Anwendungen entwickelt. Die unternehmensübergreifende Kopplung unter dem Schlagwort Business to Business Integration (B2BI) hat den Umgang mit unterschiedlichen Partnern, Datenformaten,

Technologien, Prozessen und Anforderungen zum Ziel. Mit klassischen Integrationsansätzen müssen entweder die proprietären Datenformate der einzelnen Prozesse direkt ineinander konvertiert werden, oder – bei Nutzung eines traditionellen Datenbanksystems – ein unternehmensweites oder gar unternehmensübergreifendes Datenbankschema entworfen werden. Beide Alternativen führen zu langen Entwicklungszeiten und kaum überschaubaren IT-Architekturen. Durch seine Kombination von Daten mit ihren Metadaten ermöglicht XML einen inkrementellen, bedarfsorientierten Integrationsansatz als goldenen Mittelweg. Daten aus unterschiedlichen Quellen können zunächst in ein von den innerbetrieblichen Prozessen entkoppeltes XML-Warehouse aufgenommen werden. Dort werden sie mit Hilfe etablierter XML-Standards wie XSLT, XPath und XML-Schema validiert, bereinigt, gefiltert, abgeglichen und ähnlich wie beim medienneutralen Publizieren für die verschiedenen Zielsysteme transformiert. Dieser Vorgehensweise minimiert gegenseitige Abhängigkeiten und ermöglicht eine lose Kopplung der Systeme.

Die mit dem Erfolg von XML einhergehende Durchdringung von IT-Architekturen bringt jedoch existierende XML-Werkzeuge schnell an ihre Grenzen beim Umgang mit großen Datenvolumen. Das hat vor allem zwei Gründe:

Verbosität: Der große Vorteil von XML, nämlich die Kombination von Daten mit ihren Metadaten, führt zu großen Dokumenten. In vielen Dokumenten ist die Auszeichnungsinformation größer als die ausgezeichneten Daten. Der direkte Umgang mit textuellem XML belastet daher die Speichersysteme genauso wie die zur Verfügung stehende Netzbandbreite.

Mangelnde Skalierbarkeit: Die existierenden XML-Werkzeuge werden immer wieder wegen ihres Speicherverbrauchs und ihrer Verarbeitungsgeschwindigkeit kritisiert. Dies betrifft insbesondere Realisierungen der Programmierschnittstelle Document Object Model (DOM) des W3C, die XML-Dokumente komplett als Baum im Hauptspeicher darstellen. Je nach XML-Dokument und DOM-Implementierung kann ein textuelles XML-Dokument in einem Hauptspeicher-DOM auf die zwanzigfache Größe anschwellen. Ähnliche Probleme treten bei Implementierungen der Transformationssprache XSLT auf [XSLT99].

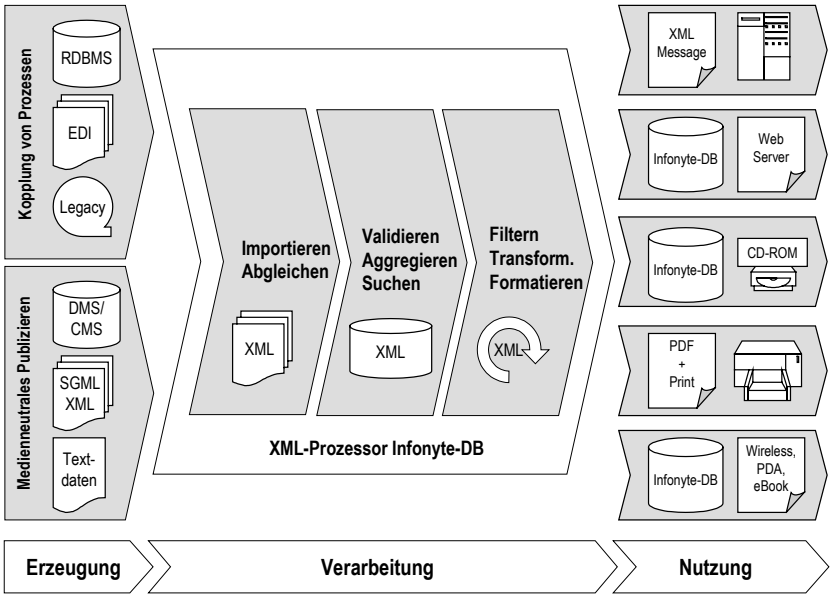


Bild 1: XML-Verarbeitung mit Infonyte-DB

Eine Möglichkeit zum Umgang mit XML-Massendaten sind native XML-Datenbanksysteme oder mit XML-Fähigkeiten angereicherte relationale Datenbanksysteme. Beide stellen die Speicherung von vielen XML-Dokumenten mit entsprechenden Suchmöglichkeiten in den Mittelpunkt. Bei der Nutzung von XML zur Integration zeigt sich jedoch, dass die skalierbare Verarbeitung weitaus stärker im Vordergrund steht als die dauerhafte Ablage. Publikationsumgebungen, die Ur-Domäne der XML-Verarbeitung, erfordern neben leistungsfähigen Autorenwerkzeugen in erster Linie die effiziente Filterung, Transformation und Formatierung zur Erzeugung verschiedener Zielformate. Auch im Bereich der Datenkonvertierung sind weniger die Ablage als vielmehr der effiziente Abgleich mit Quellen beim Import und die anschließende Aggregation und Transformation gefragt. Diese Anforderungen mit reinen XML-Speichersystemen zu realisieren scheitert oft an unzureichend integrierten DOM- und XSLT-Implementierungen. Bild 1 stellt die Nutzung von Infonyte-DB zur XML-Verarbeitung sowohl beim medienneutralen Publizieren als auch zur Kopplung von Geschäftsprozessen dar.

Der im Nachfolgenden beschriebene persistente XML-Prozessor Infonyte-DB hat sich auf den Einsatz von XML für medienneutrales Publizieren und die Integration von Geschäftsprozessen spezialisiert. Das Produkt stellt die XML-Verarbeitung in den Mittelpunkt und betrachtet die Speicherung von XML-Daten lediglich als Voraussetzung, um mit Standard-PC-Hardware auch mit Datenvolumen im Gigabyte-Bereich zu skalieren. Selbst bestehende XML-Anwendungen, die über die Standardschnittstellen DOM oder XSLT arbeiten, können durch den Austausch der

entsprechenden Module mit der Infonyte-Lösung zu skalierbaren XML-Anwendungen umgerüstet werden.

3 Infonyte-DB

Beim Entwurf von Infonyte-DB wurde das Ziel verfolgt, einen modularen und skalierbaren XML-Kern zu bauen, der sich anwendungsspezifisch erweitern lässt. Um ein hohes Maß an Integrierbarkeit zu erreichen, wurde darauf geachtet, die Kommunikation zwischen der Anwendung und dem Infonyte-Produkt soweit wie möglich über standardisierte Schnittstellen zu realisieren. Nutzer erreichen so selbst beim Einsatz einer neuen und innovativen Technologie ein hohes Maß an Investionssicherheit, da Investitionen in proprietäre Schnittstellen vermieden werden.

3.1 Überblick

Bild 2 zeigt die Architektur von Infonyte-DB. Die Architektur ist modular aufgebaut, wodurch die einzelnen Komponenten bis auf wenige Ausnahmen auch unabhängig voneinander eingesetzt werden können. Den Kern bildet ein persistentes DOM, auf dem in der XML-Welt gängige Anfrage- und Transformationsprozessoren realisiert sind. Diese nutzen für Optimierungszwecke zwar spezielle Eigenschaften des persistenten DOMs aus, können aber auch auf anderen DOM-Implementierungen aufsetzen. Für die effiziente Verarbeitung von großen XML-Dokumenten (ab ca. 500 MB) zu ermöglichen, können zusätzlich Indizes auf XML-Dokumenten (oder Kollektionen) definiert werden. Der dargestellte Kollektions-Manager bietet erweiterte Funktionen zum Umgang mit Dokumentmengen.

Das Produkt Infonyte-DB ist vollständig in Java implementiert und so auf praktisch allen marktrelevanten Plattformen ohne Probleme lauffähig. Die Entscheidung, auf Java zu setzen, hat sich als richtig erwiesen, da Java sich bei internetbasierten Serveranwendungen durchgesetzt hat und beim gezielten Einsatz und Verzicht auf ineffiziente Sprachkonstrukte in punkto Performance anderen Sprachen kaum noch nachsteht. Die verfügbaren Just-in-time-Übersetzer haben gegenüber statischen Übersetzern ohnehin den Vorteil, dass ihnen zusätzliche Laufzeitinformationen zur Verfügung stehen, die sie für Optimierungszwecke einsetzen. Die Infonyte-DB-Komponenten haben je nach Ausbaustufe eine Code-Größe zwischen 400 KB und 800 KB und sind bereits ab 16 MB RAM lauffähig.

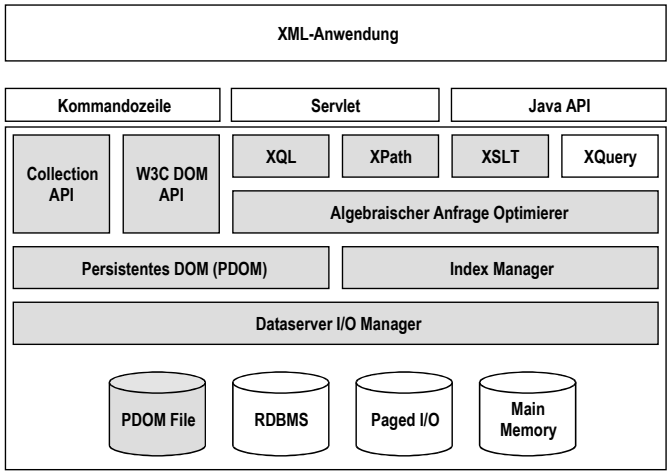


Bild 2: Infonyte-DB Komponentenarchitektur

Das System kann von einer Anwendung direkt über die Kommandozeile, einen Web-Server oder Java-Schnittstellen angesprochen werden. Es verarbeitet direkt wohlgeformtes XML, ohne dass eine DTD oder ein Schema angegeben werden muss. Alle Index- und Speicherungsstrukturen werden aus den Instanzinformationen gewonnen. Somit entfällt die aufwändige Angabe von Abbildungsvorschriften auf physische Datenmodelle, die bei der XML-Speicherung in relationalen Systemen und auch einigen XML-Datenbanken nötig sind und bei jeder Änderung der Dokumentstruktur einen großen Anpassungsaufwand nach sich ziehen.

Im Folgenden werden die einzelnen Komponenten genauer beschrieben.

3.2 Das persistente DOM

Die persistente DOM-Komponente (PDOM) bildet den Kern der Infonyte-Architektur. Das World Wide Web Konsortium (W3C) hat mit dem DOM die Standard-Programmier-Schnittstelle zum Umgang mit XML-Dokumenten geschaffen. Im DOM wird ein XML-Dokument als Baum mit unterschiedlichen Knotentypen dargestellt, welche die speziellen Eigenschaften von Elementen, Attributen etc. modellieren. So kann eine Anwendung über die Dokumentenstruktur navigieren und flexibel Knoten hinzufügen, verändern und löschen.

Es gibt heute eine Vielzahl von DOM-Implementierungen für alle gängigen Programmiersprachen. Sie bauen den Dokumentbaum in der Regel komplett im Hauptspeicher auf, was, je nach Implementierung und Hostsprache, zu erheblichen Speicherproblemen führt. Ein 20 MB XML-Dokument belegt in einem Hauptspeicher-DOM zwischen 200 MB und 400 MB.

Das Infonyte-PDOM geht hier einen anderen Weg. Es bietet eine vollständig W3C-DOM-konforme Schnittstelle, verwaltet XML-Dokumente jedoch in einem hochkompakten, verlustfreien Binärformat, dessen Speicherlayout auf die für XML spezifischen Baumstrukturen ausgelegt ist. Zur Addressierung von Dokumentknoten mit Hilfe einfacher Integer-Arithmetik, sowie zur effizienten Evaluierung von XPath-Ausdrücken, enthält das Format Indizes für Dokumentstruktur und -reihenfolge. Redundante Informationen, wie Elementnamen, werden faktorisiert. Das bewirkt, dass selbst mit den zusätzlichen Indizes, der Speicherbedarf lediglich zwischen 30% und 100% des ursprünglichen XML-Dokuments liegt. Eine Vorläuferversion des verwendeten Formats ist in [HMF99] beschrieben.

Die Kompaktheit des Binärformats sowie das für XML optimierte Speicherlayout führen zu einem hochperformanten IO-Verhalten. Der dazu eingesetzte LRU-Cache operiert auf physischen Speichersegmenten mit einer konstanten Anzahl von DOM-Knoten. Des weiteren stehen Synchronisationsmechanismen für Multithreading-Zugriffe und Funktionen zur Wartung und Defragmentierung der persistenten Dokumente zur Verfügung.

3.3 Data-Server

Das Data-Server-Modul implementiert den eigentlichen IO auf dem Speichermedium. Diese Komponente abstrahiert von den konkreten Eigenschaften eines Speichermediums und bietet wahlfreien Zugriff auf beliebig große Datensegmente. Die Data-Server-Schnittstelle ist vollständig offengelegt, so dass bei Bedarf eigene Implementierungen realisert werden können. Die mit Infonyte-DB mitgelieferte Data-Server-Implementierung verwaltet die Speichersegmente in speziell auf XML und effizienten IO zugeschnittenen Binärdateien. Wie in der Architektur angedeutet, lassen sich Data-Server-Implementierungen auch auf einer relationalen Datenbank aufsetzen oder zur Realisierung einer Hauptspeicherdatenbank verwenden [MBK00]. Genauso könnte ein Data-Server auf seine Daten über Netzdienste zugreifen, um die Skalierung im Mehrbenutzerbetrieb zu erhöhen. Die Offenheit der Backend-Schnittstelle ermöglicht die Erstellung gerätespezifischer Data-Server, so dass der Infonyte-DB leicht auf die Eigenschaften unterschiedlicher Speichersysteme vom Handheld bis zum hochskalierbaren NAT-Server angepaßt werden kann.

3.4 XML-Abfragen und XSL-Transformation mit PXSLT

Infonyte-DB unterstützt die gängigsten XML-Abfragesprachen und Transformationswerkzeuge. Damit können ein oder mehrere Dokumente durchsucht werden, die Suchergebnisse kombiniert werden, sowie logische Sichten realisiert werden. Die Extensible Query Language (XQL) ist ein Vorläufer aus dem Jahre 1999 [XQL99], wurde aber nie vom W3C als Standard verabschiedet. Heute dominieren XPath, das auch in XSLT zur Selektion von Dokumentteilen Verwendung findet, und der zukünftige XQuery-Standard die Szene [XPath99,XQuery02]. Sowohl XPath als auch XQL-Anfragen werden in der Infonyte-Architektur zunächst in einen initialen Ausführungsplan übersetzt, der in einer an XPath angelehnten Algebra dargestellt wird. Dieser Ausführungsplan wird durch die Anwendung von Transformationsregeln

optimiert, um die Ausführungszeit der Anfrage zu minimieren (logische Optimierung). Eine anschließende physische Optimierung unter Einsatz der beschriebenen Indexstrukturen führt dann zu einer optimierten Sequenz von DOM-Operationen, um das Ergebnis der Abfrage zu instanziieren.

Der Infonyte-XSLT-Prozessor („Infonyte-PXSLT“ für Persistent XSLT) bietet erstmalig die Möglichkeit, XSLT-Verarbeitung direkt auf einer persistenten Repräsentation von XML-Dokumenten durchzuführen. Gerade im Bereich der Verarbeitung großer XML-Dokumente mit XSLT gibt es massive Probleme mit den existierenden hauptspeicherbasierten Implementierungen. Dies führt in der Praxis häufig zur Stückelung der XML-Dokumente in mehrere noch handhabbare Einheiten und das Hintereinanderschalten mehrerer XSLT-Skripte. Der PXSLT-Prozessor kann aufgrund seiner Architektur ohne Schwierigkeiten XML-Dokumente im Gigabyte-Bereich mit konstantem Hauptspeicherbedarf verarbeiten. Durch die Verwendung des kompakten PDOM-Formates in Verbindung mit darauf zugeschnittenen Caching-Verfahren geschieht dies praktisch ohne Einbußen bei der Verarbeitungsgeschwindigkeit. Damit ermöglicht der PXSLT-Prozessor auch die Realisierung hochvolumiger XML-Anwendungen.

Der kommende XQuery-Standard ist in der Architektur bereits dargestellt. XQuery nutzt den XPath-Standard zur Auswahl von Dokumentfragmenten und erweitert ihn um SQL-artige Konstrukte für die Kombination und Restrukturierung von Dokumentfragmenten. Zur Darstellung von XML-Anfrageresultaten nutzt XQuery konsequenterweise XML selbst. Darüber hinaus definiert XQuery die notwendigen Funktionen und Operatoren für alle eingebauten Datentypen von XML-Schema. Ein Prototyp, der auf dem DOM-API aufsetzt, ist bereits vorhanden [FGO02].

3.5 Benutzerdefinierte Indizes

Für große XML-Dokumente und Dokumentkollektionen, an die inhaltsorientierte Anfragen gestellt werden, reichen die dargestellten Strukturindizes nicht aus. Um auch die Werte von Textknoten in einem Index zu erfassen, können benutzerdefinierte Indizes angelegt werden. Je nach Typ der zu indizierenden Werte werden Integer-, Double- oder String-Indizes erzeugt. Für textbasierte Suche ist die Erstellung eines Wortindex möglich. Indexeinträge können entweder auf die indizierten Knoten selbst oder wahlweise auch auf andere Knoten (z. B. die Dokumentwurzel) verweisen. So lassen sich beispielsweise auch Wortindizes für große Dokumentmengen aufbauen, die direkt Dokumente zurückgeben. Des Weiteren lassen sich die vorhandenen Indexstrukturen durch den Anwender erweitern, um spezielle Typen wie beispielsweise die Indizierung von Währungsformaten oder proprietären Zahlendarstellungen zu unterstützen.

3.6 Performanz

Experimente mit einer XML-Version der frei verfügbaren CD-Datenbank FreeDB [FreeDB02] sollen einen Anhaltspunkt für die Leistungsfähigkeit des Systems liefern. Bild 3 zeigt die CD-Datenbank im Administrations-GUI des Infonyte-Systems. Die

FreeDB enthält ca. 500.000 CD-Beschreibungen mit einer Datengröße von etwa 500 MB als XML-Rohtext.

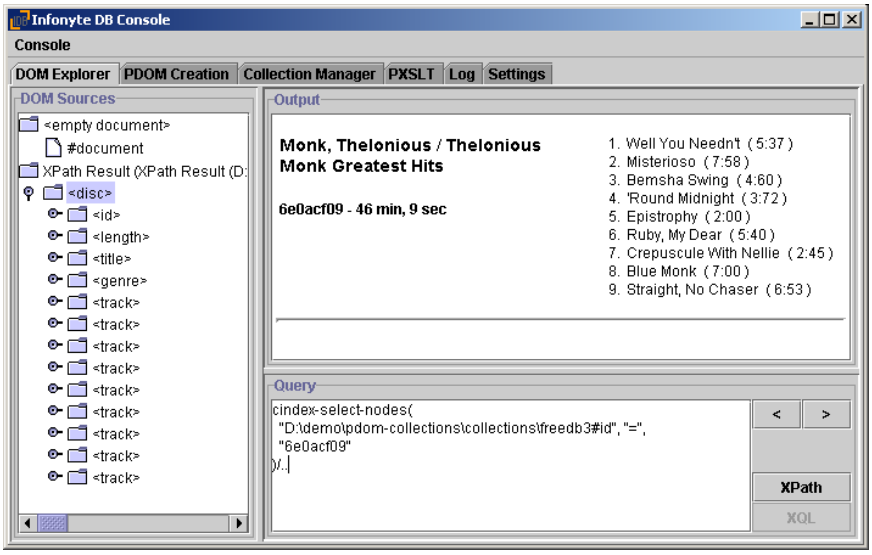


Bild 3: Screenshot der Infonyte-DB Administrationskonsole

Auf einem Standard-PC (1,8 Ghz, 512 MB RAM) dauert das Parsen und die Erzeugung eines PDOM (32 Millionen XML-Knoten, 400 MB) inklusive aller Strukturindizes etwa 4 Minuten (~2MB/Sekunde). Das Anlegen eines benutzerdefinierten Index über die einzelnen CD-Schlüssel führt zur Indizierung von 548.000 Knoten oder 1,7% der gesamten Datenbank und dauert etwa 88 Sekunden. Der Aufbau eines Volltextindex (28 Millionen Knoten, 89% der gesamten Datenbank) nimmt etwa 17 Minuten in Anspruch und resultiert in einer Größe von 90 MB. Je nach Indexdefinition werden zwischen 5 MB und 10 MB XML-Rohtext pro Sekunde indiziert. Bei der XSLT-Verarbeitung, beispielsweise zur Erzeugung von HTML, liegt der Durchsatz bei bis zu 10 MB pro Sekunde. Bei der Suche nach CDs mit bestimmten Titeln oder Tracks über den Volltextindex wird der erste Treffer bereits nach 5-10 Millisekunden geliefert. Jeder weitere Treffer wird in der gleichen Zeit ermittelt. Aufwendiger ist die Verarbeitung bei AND-Verknüpfungen, da hier die Schnittmenge mehrerer Teilergebnisse gebildet werden muss.

In einer eingeschränkten Version, bei der lediglich der Data-Server, das PDOM, sowie das Index- und das Collection-API (zusammen etwa 300 KB) verwendet werden, ist die FreeDB-Anwendung auch auf einem Pocket-PC der neueren Generation unter Realweltbedingungen lauffähig (iPAQ Pocket PC H3800 mit 64 MB Ram, 32 MB Rom, 206 Mhz ARM-Prozessor, 1GB IBM-Microdrive, Personal Java 1.2 Insignia Jeode). Unter Nutzung der Indizes liegen die Antwortzeiten bei boolescher Suche auch auf dieser eingeschränkten Plattform im Bereich von 1-2 Sekunden; die Suche nach nur einem Kriterium erfordert weniger als eine Sekunde.

3 Anwendungsszenarien

Im Folgenden wird die Nutzung von Infonyte-DB in drei Anwendungsszenarien dargestellt. Dabei wird der Einsatz des Systems zur Kopplung von Geschäftsprozessen sowie zum medienneutralen Publizieren im Bereich der technischen Dokumentation gezeigt. Ein weiteres Szenario verdeutlicht, dass Infonyte-DB sich aufgrund der Plattformunabhängigkeit und des sparsamen Umgangs mit Systemressourcen auch zum Aufbau von Informationssystemen für mobile Endgeräte eignet.

4.1 XML-Warehouse

Ein typisches Szenario für den Einsatz von XML zur Kopplung von Geschäftsprozessen ist der Betrieb eines XML-Warehouse, das Daten aus verschiedenen betrieblichen Informationssystemen in eine gemeinsame XML-Repräsentation bringt. Die von den Systemen produzierten XML-Daten werden dann über XSLT oder XQL/XPath-Anfragen zur Weiterverarbeitung wie beispielsweise der Veröffentlichung auf einem Web-Server aufbereitet.

Im vorliegenden Fall eines großen US-Dienstleisters für Finanzinformationen war es aus organisatorischen und technischen Gründen wichtig, das XML-Warehouse von den betriebswirtschaftlichen Anwendungssystemen so weit wie möglich zu entkoppeln, um einen unabhängigen Betrieb zu gewährleisten. Damit kamen Lösungen, die lediglich logische XML-Sichten bereitstellen, nicht in Frage. Auf Basis der Infonyte-Lösung wurde eine Anwendung entwickelt, die individualisierte Nachrichten verschickt und ein Web-Portal steuert, in dem Kunden ihre Handelsinformationen zeitnah abrufen können. Das Infonyte-System muß dabei täglich etwa 10 GB XML-Rohdaten aufnehmen, indizieren und in einem Fenster von zehn Tagen verfügbar halten. Es ließ sich dabei mit geringem Aufwand als skalierbares XML-Backend in den J2EE-konformen IBM WebSphere Application Server integrieren. Die Fähigkeit zur Verarbeitung der geforderten Datenmenge zusammen mit Zugriffen im Millisekundenbereich zur dynamischen Aufbereitung individualisierter Web-Seiten machen Infonyte zu einer idealen Lösung für diesen Problembereich. Mit der Automatiserung eines ehemals auf die Erzeugung und den Ausdruck von individuellen Reports ausgerichteten Geschäftsprozesses (Mainframe-Anwendung) und dem gleichzeitigen Einsatz preiswerter PC-Hardware konnten bedeutende Kosteneinsparungen erreicht werden.

4.2 Interaktive Elektronische Technische Dokumentation (IETD)

Im Bereich der technischen Dokumentation in der Luftfahrtindustrie wurde schon sehr früh SGML-Technologie eingesetzt. Aus Kostengründen werden viele Systeme heute auf XML-Technolgie und Standard-Internet-Browser umgestellt. Die Herausforderungen bestehen dabei in der Realisierung eines verteilten Autorensystems mit zentraler Datenhaltung, eines effizienten Produktionsprozesses zur Zusammenstellung und Formatierung elektronischer Handbücher für verschiedene Zielgruppen, der Bereitstellung leistungsfähiger Lese- und Navigationswerkzeuge sowie in einer möglichst kostengünstigen und sicheren Verteilung.

Aufgrund der leichten Integrierbarkeit von Infonyte und der Fähigkeit, effizient mit großen Einzeldokumenten umzugehen, hat die Sikorsky Aircraft Corporation ein XML-fähiges IETD-System auf Basis von Infonyte entwickelt. Dabei wurde sowohl der Produktionsprozess als auch die Bereitstellung der Dokumente über Web-Server mit der Infonyte-Lösung realisiert. Bei der Produktion steht die Fähigkeit, große XML-Datenmengen bedarfsgerecht zusammenzusetzen, im Vordergrund, wofür sich der Infonyte-XSLT-Prozessor bestens eignet. Bei der anschließenden Nutzung der technischen Dokumente in einer Leseumgebung kann durch die client-seitige Verwendung von Infonyte-DB über die zur Verfügung stehenden XML-Abfragesprachen zielgerichtet in bestimmten Dokumentteilen gesucht werden. Weiterhin wird die Wartung der Dokumente wesentlich erleichtert, da Service-Techniker direkt Änderungsvorschläge oder Reparaturreports über das updatefähige PDOM hinzufügen können.

Gerade im Luftfahrtbereich, wo bisher SGML dominiert hat, werden erhebliche Einsparungen bei der Entwicklung von XML-basierten IETD-Systemen erzielt. Zum einen kann Client-seitig vielfach Standardsoftware (XML/PDF Browser) eingesetzt werden, so dass aufwendige Eigenentwicklungen entfallen, und zum anderen ist der gesamte Entwicklungsprozess durch den Einsatz von XML-Werkzeugen wesentlich günstiger.

Ähnliche Einsatzszenarien gibt es im Bereich formularorientierter Verarbeitungsprozesse und der Aufbereitung von Fachinformationstexten und Nachschlagewerken im medizinischen und juristischen Bereich.

4.2 Mobiles Informationsmanagement

Der geringe Speicherverbrauch von Infonyte-DB, die Plattformunabhängigkeit durch Java sowie die Kompaktheit des PDOM-Formats machen Infonyte zu einer idealen Lösung für XML-basiertes mobiles Informationsmanagement der nächsten Generation. Die Firma Vaultus (http://www.vaultus.com) entwickelt unter anderem auf Basis der Infonyte-Technologie eine mobile Informationsplattform. Neben dem Datenmanagement und Nachrichtenaustausch in XML bietet das System Offline-Fähigkeiten, sichere Transaktionen, Netzwerkunabhängigkeit und Dienste zur Fernwartung und automatischen Aktualisierung mobiler Geräte. Der Infonyte-XML-Prozessor kommt hier aufgrund seines sparsamen Umgangs mit Systemressourcen ausschließlich auf den Java-fähigen Mobilgeräten zum Einsatz. Die XML-Daten werden ähnlich wie bei Nutzung einer relationalen Datenbank direkt von der Anwendung manipuliert, d. h. Updates auf der feingranularen Dokumentenstruktur wie das Infonyte-PDOM sie bietet sind eine Schlüsselfähigkeit zum Einsatz in dieser Anwendungsklasse. Für zukünftige Anwendungen ist auch der Einsatz des kompakten PDOM-Formats zum direkten Austausch von XML-Daten für Synchronisationszwecke denkbar. Das Parsen von Dokumenten könnte dann vollständig entfallen und sämtliche Daten wären sofort über die kodierten Indexstrukturen effizient zugreifbar. Bild 4 zeigt den Screenshot einer mobilen Vertriebsanwendung auf Basis der Vaultus-Plattform.

Bild 4: Web-Demo einer Mobilanwendung mit Infonyte-DB

4 Ausblick

Skalierbare XML-Prozessoren wie Infonyte-DB zeigen einen neuen innovativen Ansatz auf, die Probleme beim Umgang mit großen XML-Daten und damit die Skalierungsprobleme bestehender Werkzeuge zu überwinden. Eigene Experimente mit Open-Source-Implementierungen zeigen, dass viele freie Realisierungen (FOP-Prozessoren, XML-Editoren, WebDAV-Server) durch Austausch des Speicher-Backend mit der Infonyte-Lösung zur skalierbaren XML-Anwendung aufgerüstet werden können. Die Beschränkung auf einen schlanken XML-Kern sowie die performante Realisierung in einer plattformunabhängigen Sprache wie Java machen Infonyte zu einem nahezu universell einsetzbaren Werkzeug für XML-Anwendungen. Der große Bedarf, existierende Architekturen um XML-Fähigkeiten zu ergänzen, zusammen mit der leichten Integrierbarkeit machen das System auch zur kostengünstigen Erweiterung bestehender Anwendungen interessant.

Infonyte-DB kann zur kostenlosen Evaluierung unter der Adresse http://www.infonyte.de heruntergeladen werden.

Literaturverzeichnis

[BeMW02] Beimborn, Daniel; Mintert, Stefan; Weitzel, Tim: Web Services und ebXML. Erscheint in: WIRTSCHAFTSINFORMATIK (2002) 3.

[BuLW01] Buxmann, Peter; Ladner, Frank; Weitzel, Tim: Anwendung der Extensible Markup Language (XML): Konzeption und Implementierung einer WebEDI-Lösung, In: WIRTSCHAFTSINFORMATIK (2001) 3, S. 257-267.

[DOM00] Arnaud Le Hors et al.: Document Object Model (DOM) Level 2 Core Specification. W3C Recommendation. 13. November 2000, http://www.w3.org/TR/DOM-Level-2-Core/, Abruf am 2002-06-06.

[FGO02] Fankhauser, P.; Groh, T.; Overhage, S.: XQuery by the Book: The IPSI XQuery Demonstrator. EDBT 2002: 742-744. http://xml.darmstadt.gmd.de/xquerydemo/ , Abruf am 2002-06-06.

[FreeDB02] http://www.freedb.org, Abruf am 2002-06-06.

[HMF99] Huck, G.; Macherius,I.; Fankhauser, P.: PDOM: Lightweight Persistency Support for the Document Object Model. OOPSLA Workshop "Java and Databases: Persistence Options". November 1999. Denver, Colorado, USA.

[Luo01] Luoma, R.: Using XML to Enable Low-Cost Deployment of Content at Lockheed Martin Aeronautics. XML 2001 Conference and Exposition. December 2001. Orlando, Florida, USA.

[MBK00] Manegold, S.; Boncz, P.A.; Kersten, M.L: Optimizing database architecture for the new bottleneck: memory access. VLDB Journal (2000) 9(3), S. 231-246.

[RaAS02] Rawolle, J.; Ade, J.; Schumann, M: XML als Integrationstechnologie bei Informationsanbietern im Internet - die Fallstudie BertelsmannSpringer, In: WIRTSCHAFTSINFORMATIK (2002) 1, S. 19-28.

[Nwg99] Network Working Group: HTTP Extensions for Distributed Authoring – WEBDAV. RFC 2518. http://www.ietf.org/rfc/rfc2518.txt, Abruf am 2002-06-06.

[XML98] Tim Bray et al.: Extensible Markup Language (XML) 1.0. W3C Recommendation. 10. Februar 1998. http://www.w3.org/TR/1998/REC-xml-19980210.html, Abruf am 2002-06-06.

[XQL99] Robie, J.; Lapp, J.; Schach, D.: XML Query Language: A Proposal. W3C-QL '98 workshop proposal. http://www.w3.org/Style/XSL/Group/1998/09/XQL-proposal.html, Abruf am 2002-06-06.

[XQuery02] Draper, D. et al.: XQuery 1.0 Formal Semantics. W3C Working Draft. 26. März 2002. http://www.w3.org/TR/query-semantics/, Abruf am 2002-06-06.

[XPath99] Clark, James et al.: XML Path Language (XPath) Version 1.0. W3C Recommendation. 16 November 1999. http://www.w3c.org/TR/xpath/, Abruf am 2002-06-06.

[XSD01] Fallside, David C.: XML Schema Part 0: Primer. W3C Recommendation, 2. Mai 2001. http://www.w3.org/TR/xmlschema-0/, Abruf am 2002-06-06.

[XSLT99] Clark, James: XSL Transformations (XSLT) Version 1.0. W3C Recommendation. 16. November 1999. http://www.w3c.org/TR/xslt/, Abruf am 2002-06-06.

Manipulation von XML-Dokumenten in Tamino

Dr. Michael Gesmann
Software AG, Darmstadt
michael.gesmann@softwareag.com

Abstract: Dieser Beitrag untersucht die Frage, wie XML-Dokumente, die in Datenbanksystemen abgespeichert sind, effektiv und effizient innerhalb dieser Systeme verändert werden können. Zunächst wird skizziert, welche Lösungsansätze bereits existieren. Danach wird erläutert, wie dieses Problem in der neuen Version des XML-Datenbanksystems Tamino von der Software AG gelöst wird. Dort wird ein sprachbasierter Ansatz auf der Basis von XQuery-Erweiterungen verfolgt.

1 Die Aufgabe

Mittlerweile ist unbestritten, dass auch XML-Dokumente in Datenbanken verwalten werden sollen. Auch die größten Datenbankhersteller arbeiten an effizienten und benutzerfreundlichen Schnittstellen zur Abspeicherung und Bearbeitung solcher Dokumente.
Als Anfragesprache erarbeitet das W3C zur Zeit eine Empfehlung für XQuery [XQ02]. Jede andere Funktionalität ausser Anfragen wurde dort explizit herausgelassen, um den Definitionsprozess nicht unnötig zu verlängern. Dies gilt auch für Änderungsoperationen in XML-Dokumenten. Für die XML-Technologie gilt daher zur Zeit, dass es keine dem SQL-Update vergleichbare standardisierte Operation gibt.
Anwendungen, die in einer Datenbank abgespeicherte XML-Dokumente verändern wollen, bleibt damit nur die Manipulation der Dokumente in Anwendungsprogrammen. Dokumente können mittels XQuery gelesen, dann von einer Anwendung verändert und abschließend wieder über die jeweilige Systemschnittstelle in die Datenbank zurückgeschrieben werden. Dabei wird die Performanz in vielen Fällen völlig inakzeptabel. Wenn zum Beispiel in einem Bericht ein einzelner Abschnitt eingefügt, entfernt oder einfach nur ein Rechtschreibfehler korrigiert werden soll, muss dazu der gesamte Bericht ausgelesen und abgespeichert werden. Der Dokumenttransfer von der Datenbank zur Anwendung und zurück verursacht viel zu hohen Zusatzaufwand, da viele nicht benötigte Teile des Dokuments auch gelesen und geschrieben werden müssen. Die schlechte Performanz zwingt alle Datenbankhersteller, zusätzliche Funktionalität bereitzustellen, mit der Dokumentänderungen direkt in der Datenbank möglich werden. Weil ein einheitlicher Standard fehlt und Lösungen schnell benötigt werden, entstehen viele verschiedene Lösungen.
Im folgenden Abschnitt werden bereits existierende Lösungen skizziert. Der Ansatz, auf den wir bei der Entwicklung von Tamino zurückgreifen, wird in den folgenden Abschnitten ausführlicher beschrieben. Grundlegende Kenntnisse über XML, das XQuery-Datenmodell und XQuery selber werden hier vorausgesetzt.

2 Lösungsansätze und eine grobe Bewertung

Während über Anfragesprachen für XML und die Abspeicherung von XML in den vergangenen Jahren eine Vielzahl an Publikationen erschienen ist, wurden Änderungen von XML-Dokumenten in einer Datenbank bisher sehr wenig untersucht. In diesem Abschnitt sollen einige Ansätze und Untersuchungen beschrieben und bewertet werden, soweit dies zur Zeit überhaupt möglich ist.

Bewertungskriterien
Für die Bewertung werden folgende wesentliche Kriterien herangezogen:

- Performanz: Ein Ansatz sollte einer möglichst großen Anzahl von Anwendungen eine gute Performanz ermöglichen. Dabei sollte die Performanz nicht allein auf effizienter Anwendungsimplementierung beruhen, sondern auch auf die Optimierungsmöglichkeiten eines Datenbanksystems zurückgreifen können.
- Ausdrucksstärke: Der zu realisierende Mechanismus sollte einer breiten Anwendungspalette die Möglichkeit geben, Änderungsoperationen zu spezifizieren. Dazu muss mindestens folgende Basisfunktionalität angeboten werden:
  - Navigation zu Knoten, deren Inhalte tatsächlich geändert werden sollen
  - Berechnung neuer Knoteninhalte
  - Einfügen neuer Knoten in ein Dokument
  - Löschen von Knoten im Dokument
  - Umhängen bereits existierender Knoten an eine andere Stelle im Dokument
  - Ändern von Knoteninhalten, zum Beispiel Text in einem Textknoten
  - Umbenennen von Knoten.

  Außerordentlich hilfreich sind folgende Eigenschaften:
  - Ausführung mehrerer Basisfunktionen in einer Änderungsoperation
  - Schachtelung von Anfragen in die Änderungsoperationen
  - Temporäre Akzeptanz von Dokumenten, die gemäss einem gegebenen XML-Schema ungültig oder sogar nicht wohlgeform sind

  Im Datenbanksystem ist außerdem wünschenswert:
  - Das Sichtfeld einer Änderungsoperation sollte einzelne Dokumente übersteigen, das heißt, für Änderungen in einem Dokument "A" sollten auch Inhalte anderer Dokumente in der Datenbank mit einbezogen werden können (vgl. Join-Funktionalität in XQuery).
- Einfachheit: Eine Sprache zur Beschreibung von Änderungsoperationen sollte für Benutzer möglichst einfach verständlich und anwendbar sein.

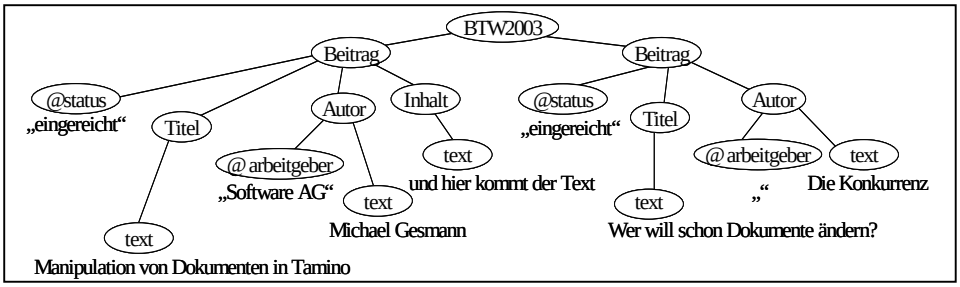
Weiterhin sollte jeder Ansatz mit anderen XML Empfehlungen des W3C konform sein. Ansätze, die zum Beispiel mit den Empfehlungen für XPath [XP99] oder Namespaces [XNs99] nicht zu vereinbaren sind, werden nur als proprietäre und schwer vermittelbare Speziallösungen Bestand haben können.
Die Forderung aus den XUpdate-Anforderungen von XML:DB [Xup02R], dass die Formulierung in Form eines XML-Dokuments erfolgen muss, wird hier explizit weggelassen. Die Erfahrungen mit XQuery zeigen, dass es sehr schwer ist, eine übersichtliche und einfach verständliche XML-Syntax für eine solche Sprache zu definieren.

Beispiel
Anhand des folgenden einfachen XML-Dokuments sollen Änderungsoperationen illustriert werden:

```
<?xml version="1.0"?>
<BTW2003>
  <Beitrag status="eingereicht"> <Titel>Manipulation von Dokumenten in Tamino</Titel>
                                 <Autor arbeitgeber="Software AG">Michael Gesmann</Autor>
                                  <Inhalt> und hier kommt der Text </Inhalt>          </Beitrag>
  <Beitrag status="eingereicht"> <Titel>Wer will schon Dokumente ändern?</Titel>
                                 <Autor arbeitgeber="">Die Konkurrenz</Autor>       </Beitrag>
</BTW2003>
```

Dokument 1: Ausgangsdokument zur Erläuterung von Beispielen

Zur Illustration von durchzuführenden Änderungsoperationen ist es häufig günstiger, das Dokument als Baum aufzuzeigen:



In den nachfolgend skizzierten Ansätzen soll jeweils der Titel des Beitrags vom Autor "Michael Gesmann" auf "Änderung von Dokumenten in DBMS" modifiziert werden.

2.1 Offen gelegte Datenbankfunktionalität

Die XML-Erweiterungen der relationalen Datenbanksyteme von IBM und Oracle ermöglichen derzeit Änderungen "lediglich" über vorgegebene Schnittstellen von UDFs (user defined functions) an[1].

Update() von DB2
Die "update(Dokument, LocationPath, Wert)"-Funktion auf Spalten vom Typ "XML UDT" in [Db2] bekommt drei Argumente; zuerst das zu verändernde Dokument, dann einen LocationPath, über den eine Menge von Knoten in dem Dokument bestimmt wird, und als drittes noch einen Wert, der in allen gefundenen Knoten deren alten Wert ersetzt. Eine Änderungsoperation sieht hier folgendermaßen aus:

[1]Dokumente können in diesen Systemen auch in Tabellen und Spalten strukturiert werden anstatt sie in Textform in einem CLOB abzulegen. Dann bleibt als besonders effizienter Weg auch die Möglichkeit, die generierten SQL-Strukturen direkt zu manipulieren. Dieser Weg wird hier nicht weiter berücksichtigt, weil er keine echte Alternative darstellt. Dieser Ansatz gewährleistet nicht ohne weiteres, dass die Dokumente gemäß einem gegebenen Schema gültig bleiben. Ausserdem führen Änderungen im XML-Schema zu veränderter Abbildung auf Tabellen, wodurch Anwendungen teilweise unnötigerweise überarbeitet werden müssen.

```
UPDATE TabelleMitTagungsDokumenten
SET dokumentSpalte = db2xml.update(dokumentSpalte,
                          '/BTW2003/Beitrag[Autor/@name="Michael Gesmann"]/Titel',
                          "Manipulation von Dokumenten in DBMS")
WHERE tagungSpalte="BTW" and jahrSpalte=2003
```

Änderung 1: Beispieländerung in DB2

Prädikate im LocationPath kann DB2 nur auf Attributen auswerten. Deshalb ist eine Änderung des Prädikates und eine andere Modellierung der Dokumente erforderlich.

UPDATEXML() von Oracle

[Ora9i2] erlaubt die Abspeicherung von XML-Dokumenten in Tabellen oder Spalten vom Typ XMLType. Dokumente können in ihrer ursprünglichen Form als CLOBs abgespeichert werden. Da das System in diesem Fall die Struktur der Dokumente nicht kennt, können diese Dokumente nur in ihrer Textform ganz oder gar nicht verändert werden. Alternativ können Dokumente strukturiert abgelegt werden, wenn ein XML-Schema vorliegt. In diesem Fall können Änderungen mittels der XML member-Funktion "UPDATEXML(Dokument, Paare von LocationPath und Wert)" verändert werden. Die Schnittstelle hier ist etwas mächtiger als die von DB2. Es können mehrere XPath-Ausdrücke mit neuen Werten angegeben werden. Ferner dürfen sich Prädikate in den XPath-Ausdrücken auch auf Elemente beziehen. Somit kann eine Änderung wieder auf dem ursprünglichen Dokument ausgeführt werden:

```
UPDATE TabelleMitTagungsDokumenten
SET dokumentSpalte = UPDATEXML(dokumentSpalte,
                          '/BTW2003/Beitrag[Autor="Michael Gesmann"]/Titel',
                          "Manipulation von Dokumenten in DBMS")
WHERE tagungSpalte="BTW" and jahrSpalte=2003
```

Änderung 2: Beispieländerung in Oracle

UPDATEXML() beschreibt nur die Veränderung eines gegebenen Dokuments. Damit die Änderung auch in die Datenbank einfließt, muss der Aufruf genauso wie bei DB2 in eine SQL Update-Anweisung eingebunden werden.

Server Extensions von Tamino

Tamino [Tam41] bietet einen noch offeneren Mechanismus an, die sogenannten Server Extensions. Eine Server Extension wird als Anwendungsfunktion implementiert. Nachdem sie von einem Administrator in einer Datenbank registriert wurde, kann sie genauso wie vordefinierte Funktionen aufgerufen und im Server ausgeführt werden. Eine Server Extension kann beispielsweise zunächst eine XQuery ausführen, anschließend die Ergebnisdokumente modifizieren und schließlich wieder zurückschreiben.

Da die Funktionen in allen drei Fällen innerhalb des DBMS ausgeführt werden, entfallen Kommunikationskosten mit der Anwendung. Man kann also von diesen Ansätzen eine bessere Performanz als bei der anwendungsseitigen Verarbeitung erwarten. Bei den relationalen Erweiterungen ist die angebotene Funktionalität aber sehr stark eingeschränkt. So werden neue Werte wieder absolut vorgegeben und relative Änderungen wie das Einfügen eines neuen zusätzlichen Knotens werden nicht angeboten. Relationale Anfragen werden hier mit XPath-Ausdrücken gemischt. Die

Server Extensions von Tamino dagegen erfordern komplexe Anwendungsprogrammierung auf den Dokumenten. Man kann erwarten, dass einfachere Funktionalität auch effektiver optimiert werden kann. Auf zusätzliche Problem wie Zugriffskontrolle und Anfrageoptimierung soll hier nicht weiter eingegangen werden.

2.2 Eigenständige Beschreibungssprachen

Eine weitere Klasse von Lösungsansätzen entwickelt eigenständige Sprachen zur Beschreibung von Änderungsoperationen. Auf diese Sprachen wird als nächstes eingegangen.

XML:DB
Die XML:DB Initiative publiziert zur Zeit gleich zwei Sprachvorschläge. SiXDML (Simple XML Data Manipulation Language) [SiXDML02] ist eng an SQL angelehnt, und umfaßt auch viele Administrations- und Schemaoperationen. Im Gegensatz dazu konzentriert sich XUpdate [Xup00] auf die Manipulation von Dokumentinhalten. Änderungsoperationen werden dort in XML-Dokumenten formuliert. Für die Auswahl der zu verändernden Knoten benutzt XUpdate XPath-Ausdrücke. Elemente in einem XUpdate-Dokument beschreiben Änderungen, die in einem gegebenen Dokument ausgeführt werden sollen. Es können mehrere solcher Änderungen angegeben werden, die dann sequentiell ausgeführt werden. Die Spezifikation legt nicht fest, wie ein einzelnes oder mehrere zu verändernde Dokumente bestimmt werden.
Die Beispieländerung wird hier folgendermaßen beschrieben:

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate">
  <xupdate:update select='/BTW2003/Beitrag[Autor="Michael Gesmann"]/Titel'>
    Änderungen von Dokumenten in XML-Datenbanken
  </xupdate:update>
</xupdate:modifications>
```

Änderung 3: Beispieländerung mit XUpdate

Updategrams in eXcelon
Updategrams in eXcelons Information Server [XIS312] umfassen eine gegenüber XUpdate erweiterte Funktionalität. Dort können Teilbäume eines Dokuments kopiert werden, Änderungsanweisungen lassen sich ineinander schachteln, und mit Hilfe von Variablen und Schleifenkonstrukturen ("foreach") können auch komplexere Update-Anwendungen definiert werden. Auch Updategrams beschreiben nur, wie einzelne Dokumente geändert werden. Um eine Menge von Dokumenten zu verändern, müssen zusätzliche Mechanismen herangezogen werden.

Updategrams in Microsofts SQL-Server
Im Unterschied zu eXcelons Updategrams geben Updategrams im SQL Server von Microsoft [SQLServ2000] keine Berechnungsvorschriften an, mittels derer die Änderungen durchgeführt werden sollen. Stattdessen beschreiben Before- und After-Elemente eines Updategrams die durchzuführenden Änderungen. Ist kein Before-

Element, sondern nur ein After-Element angegeben, dann wird ein neues Dokument eingefügt. Ist nur ein Before-Element angegeben, aber kein After-Element, dann wird das im Before-Element identifizierte Dokument gelöscht. Sind sowohl ein Before-Element als auch After-Element angegeben, dann wird das im Before-Element identifizierte Dokument entsprechend den Beschreibungen im After-Element geändert. Die beschriebenen Änderungen werden in SQL-Befehle umgesetzt, mit denen die zugrundeliegenden Tabellen modifiziert werden. Wenn für die Abbildung von XML-Elementen und -Attributen auf Tabellen und Spalten kein Default-Verfahren ausreicht, muss ein Schema mit entsprechenden Informationen angegeben werden. Für die selektive Veränderung einzelner Elemente wird ein eindeutiger Identifikator benötigt. Inhalte müssen aber immer vollständig angegeben werden. Es ist zum Beispiel nicht möglich, in einen Knoten, der bereits mehrere Kinder hat, einen weiteren Knoten einzeln einzufügen. Vielmehr muss der Knoten mit seinem gesamten Inhalt neu angegeben werden.

2.3 Spracherweiterungen für XQuery

XUpdate und Updategrams sind unabhängig von der neuen Anfragesprache XQuery entstanden. Mit der Entwicklung von XQuery besteht nun auch die Möglichkeit Änderungsoperationen in XQuery zu integrieren. So beschreibt [TIHW01] zunächst eine kleine Menge von einfachen Änderungsoperationen und wie diese Operationen in einer Erweiterung von XQuery ausgedrückt werden können. Anschließend wird untersucht, mit welchen Mitteln die Operationen bei einer Abbildung der Dokumente auf relationale Tabellen ausgeführt werden können. [Le01] beschreibt einen zu [TIHW01] leicht erweiterten Funktionsumfang, der ebenfalls als Erweiterung von XQuery formuliert wird. Die in der Arbeit beschriebenen Spracherweiterungen wurden in einen XQuery Prototypen namens QuiP [QuiP00] integriert. Nach [Le01] wird die Beispieländerung folgendermaßen formuliert ([TIHW01] hat lediglich eine etwas andere Syntax):

```
update
replace input()/BTW2003/Beitrag[Autor="Michael Gesmann"]/Titel/text()
        with "Manipulation von Dokumenten in DBMS"
```

Änderung 4: Beispieländerung nach [Le01]

oder alternativ:

```
update
for $text in input()/BTW2003/Beitrag[Autor="Michael Gesmann"]/Titel/text()
replace $text with "Manipulation von Dokumenten in DBMS"
```

Änderung 5: Alternative Formulierung der Beispieländerung nach [Le01]

[TIHW02] und [Le01] decken die in den Bewertungskriterien aufgeführte Basisfunktionalität ab und erlauben es, mehrere Änderungsoperationen in einer Anfrage anzugeben. Der Anfrageteil bestimmt die zu verändernden Dokumentteile, ein Update-Teil spezifiziert die durchzuführenden Änderungen. Für die verschiedenen Suchanteile der Operationen wird immer XQuery selbst verwendet.

2.4 Zusammenfassung und weitere Aspekte

Die anwendungsseitige Verarbeitung von Änderungsoperationen ist die einzige Möglichkeit, Dokumente zu verändern, wenn das Datenbanksystem die XML-Semantik der Dokumente nicht versteht. Alle in diesem Abschnitt aufgeführten Ansätze „verstehen XML“ und wurden in DBMS implementiert, so dass die Kommunikationskosten zwischen Anwendung und DBMS reduziert werden können. Die relationalen Erweiterungen fallen vor allen Dingen durch die deutlich eingeschränkten Funktionalitäten auf. Die Beschreibung von Änderungsoperationen in Updategrams liefert eine deutlich mächtigere Funktionalität. Dort fehlt eine enge Einbindung in die mittlerweile vorhandene Anfragesprache XQuery. Dieser Punkt ist wichtig, weil die Suche der zu verändernden und der neu einzutragenden Knoten wesentlicher Bestandteile jeder Änderungsoperation ist. Der zuletzt aufgeführte Ansatz über Spracherweiterungen von XQuery, weist diese Schwäche nicht auf. Gleichzeitig erfüllt er Kriterien nach Performanz und Ausdrucksstärke mindestens genauso gut wie die anderen Ansätze. Aus diesen Grüunden wurde Spracherweiterungen von XQuery als Basis für die Weiterentwicklung von Tamino ausgewählt. Die folgende Tabelle soll die Bewertung noch einmal illustrieren und für jeden aufgeführten Ansatz festhalten, wo Stärken und Schwächen liegen („+“ verdeutlicht Stärken, „-“ deutet auf Schwächen hin):

Ansatz	Performanz	Ausdrucksstärke	Einfachheit
UDF von DB2	kann Optimierungen des DBMS nutzen (+)	Nur sehr eingeschränkte einzelne Wertveränderungen in einzelnen Dokumenten (-)	XPath, Dokumentzugriff via SQL (+/-)
UDF von Oracle9i Version 2	s.o. (+)	Ebenfalls sehr eingeschränkter Funktionsumfang (mehr als DB2), Änderungen eingeschränkt auf einzelne Dokumente (-)	XPath, Dokumentzugriff via SQL (+/-)
Server Extension von Tamino	schwer in Optimierung zu integrieren (+/-)	Anwendungsprogrammierung (+/-)	XQuery incl. XPath und DOM werden typischer-weise benötigt (-)
XML:DB - XUpdate	kann Optimierungen des DBMS nutzen (+)	XPath, Änderungen eingeschränkt auf einzelne Dokumente, mehrere Änderungsanweisungen in einer Anforderung möglich (+/-)	XPath plus Änderungsoperationen und Konstruktoren, zusätzliche Einbindung in Queryfunktionalität notwendig (+/-)
Updategram von eXcelon	s.o. (+)	Wie XUpdate aber reichhaltigerer Funktionsumfang (+/-)	Einbindung in Queryfunktionalität notwendig (+/-)
Updategram von SQLServer	s.o. (+)	Einfügen und Löschen möglich, alle anderen Operationen erfordern vollständig neue Definition des Inhalts (-)	Abbildung auf relationale Tabellen muss beachtet werden (-)
Erweiterungen von XQuery	s.o. (+)	Volle XQuery-Mächtigkeit auch über mehrere Dokumente, mehrere Anweisungen in einer Anforderung möglich (+)	XQuery plus einige einfache Änderungs-operationen und Konstruktoren (+)

Tabelle 1: Zusammenfassende Bewertung der Ansätze

3 Spracherweiterungen von XQuery in Tamino

Mit der Version 4.1 von Tamino werden Anwendungen verbesserte Mechanismen zur Modifikation von Dokumenten in der Datenbank zur Verfügung gestellt. Grundlegende XQuery Funktionalität steht in der Version 4.1 dafür zur Verfügung. Mit [Le01] war einige Grundlagenarbeit bereits durchgeführt, auf deren Basis die weitere Entwicklung angegangen wurde. Syntax und Semantik der in Tamino implementierten XQuery-Spracherweiterungen werden in diesem Abschnitt vorgestellt.
Jede **Änderungsoperation** beginnt mit dem Schlüsselwort "update". Darauf folgen eine oder mehrer UpdateExprs. Jede UpdateExpr kann entweder direkt eine der **Elementaroperation** Einfügen, Löschen, Umbenennen oder Ersetzen beschreiben, oder sie kann komplexer zusammengesetzt sein und dazu in einer FLWUExpr formuliert werden. Mit den Elementaroperationen läßt sich die gesamte in Abschnitt 2 aufgeführte Basisfunktionalität realisieren. Eine FLWUExpr sieht fast genauso aus wie eine normale XQuery, lediglich die result-Klausel einer XQuery wird durch Änderungsanweisungen ersetzt.
Abbildung 1 zeigt alle Syntaxerweiterungen. Syntaxelemente, die im XQuery Working Draft [XQ02] definiert sind, werden kursiv wiedergegeben.

```
Query           ::= QueryProlog ( [ ExprSequence] | 'update' UpdateSequence ) 'Eof'
UpdateSequence  ::= UpdateExpr +
UpdateExpr      ::= InsertClause |DeleteClause |RenameClause |ReplaceClause |FLWUExpr

InsertClause    ::= 'insert' Expr ('following'|'preceding'|'into'|) Expr
DeleteClause    ::= 'delete' Expr
RenameClause    ::= 'rename' Expr 'as' QName
ReplaceClause   ::= 'replace' Expr 'with' Expr

FLWUExpr        ::= ( ForClause | LetClause )* [ WhereClause ]
                        'do' ( UpdateExpr | Lpar UpdateSequence Rpar )
```

Abbildung 1: Syntaxerweiterungen für XQuery

Die vorhandendene XQuery Funktionalität wird für zwei Teilaufgaben benötigt. Zunächst einmal müssen diejenigen Knoten bestimmt werden, an denen Änderungen durchgeführt werden sollen. Dazu können einfache oder durch komplexere Ausdrücke mit For-, Let- und Where-Klauseln von XQuery genutzt werden. Die so bestimmten Knoten werden im folgenden **Änderungsknoten** genannt.
Zusätzlich werden die eigentlichen Änderungen beschrieben. Dabei müssen die neuen Werte keine Konstanten sein, sondern können ebenfalls durch XQuery-Ausdrücke berechnet werden. Diese Knoten, die neue Werte repräsentieren, werden im folgenden **Werteknoten** genannt.

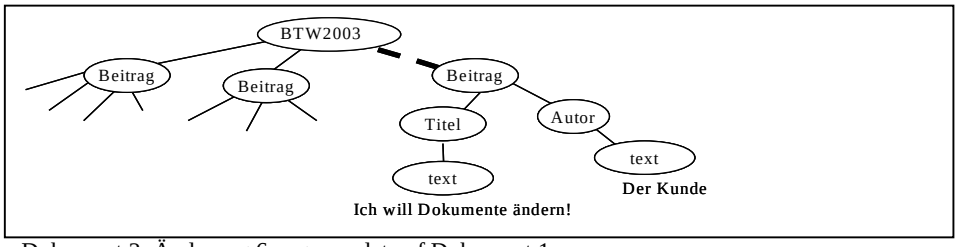
3.1 Elementaroperationen

Insert
Die Einfügeoperation dient dazu, Werteknoten an eine bestimmte Position in ein Dokument einzutragen. Mit der ersten Expr in der InsertClause werden einer oder eine Sequenz einzufügender Werteknoten bestimmt. Mit der zweiten Expr in der InsertClause

werden die Elternknoten als Änderungsknoten und die Einfügeposition bestimmt. Abhängig von dem dazwischen liegenden Schlüsselwort wird die Einfügeoperation ausgeführt.
Bei einem "insert ... into ... " werden die Werteknoten an das Ende der aktuellen Kinderliste von jedem Änderungsknoten angehängt, wenn es sich bei den Werteknoten nicht um Attribute handelt. Falls ein Änderungsknoten noch keine Kinder hat, wird die Kinderliste neu angelegt. Die Reihenfolge der Werteknoten bleibt erhalten. Wenn es sich bei den Werteknoten um ein oder mehrere Attribute handelt, dann werden diese bei einem "insert ... into ..." in den Änderungsknoten eingetragen.
Auch bei einem "insert ... preceding ..." werden die Werteknoten in den Elternknoten eingefügt. Wiederum ist der Elternknoten der Änderungsknoten. Die zweite Expr in der InsertClause charakterisiert die Eltern-Kind-Beziehung vor die unmittelbar die neuen Kinder eingefügt werden sollen. Im Fall von "insert ... following ... " werden die Werteknoten in den Elternknoten unmittelbar hinter die durch die zweite Expr charakterisierte Eltern-Kind-Beziehung eingefügt. Da es auf Attributen keine Reihenfolge gibt, sind diese beiden Formen des Einfügens nicht mit Attributen möglich.
Es folgen einige Beispiele, die auf der Basis von Dokument 1 aus Abschnitt 2 ausgeführt werden. Zuerst wird ein neuer Beitrag an das Ende des Knotens BTW2003 eingetragen:

```
update insert <Beitrag><Titel>Ich will Dokumente ändern!</Titel>
                        <Autor>Der Kunde</Autor></Beitrag>
            into input()/BTW2003
```

Änderung 6: Neues Dokument einfügen

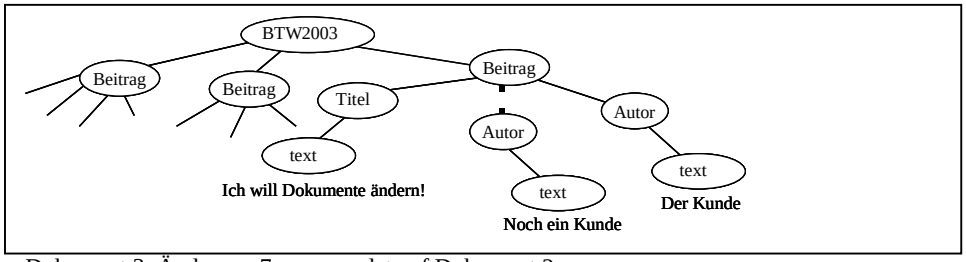


Dokument 2: Änderung 6 angewendet auf Dokument 1

Trage in das soeben neu eingefügte Beitrag-Element einen neuen Autor an den Anfang der Autorenliste ein:

```
update insert <Autor>Noch ein Kunde</Autor>
            preceding input()/BTW2003/Beitrag[Titel="Ich will Dokumente ändern!"]/Autor[1]
```

Änderung 7: Zusätzlichen Autor einfügen



Dokument 3: Änderung 7 angewendet auf Dokument 2

Trage in in den neuen Beitrag noch das Attribute „status“ ein:

```
update insert attribute status {"eingereicht"}
        into input()/BTW2003/Beitrag[Titel="Ich will Dokumente ändern!"]
```

Änderung 8: Zusätzliches Attribut einfügen

Diese drei Einfügeoperationen angewandt auf Dokument 1 (abzüglich einiger Kürzungen) führen zu folgendem Dokument:

```
<?xml version="1.0"?>
<BTW2003> <Beitrag status="eingereicht"> ... gekürzt ... </Beitrag>
          <Beitrag status="eingereicht"> ... gekürzt ... </Beitrag>
   <Beitrag status="eingereicht"> <Titel>Ich will Dokumente ändern!</Titel>
        <Autor>Noch ein Kunde</Autor> <Autor>Der Kunde</Autor></Beitrag>
</BTW2003>
```

Dokument 4: Änderungen 6 bis 8 angewandt auf Dokument 1

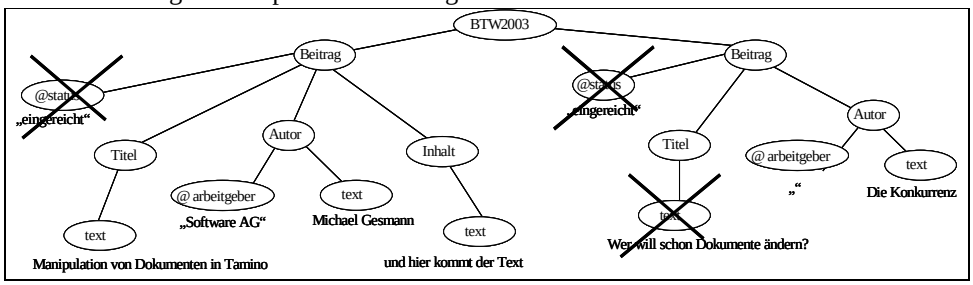
Delete

Mit Hilfe der Löschoperation können Knoten aus Dokumenten entfernt werden. Mit der angegebenen Expr werden die zu löschenden Knoten (Änderungsknoten) berechnet.
Die folgende Anweisung löscht aus allen Beiträgen das status-Attribut und löscht zusätzlich den Inhalt des Titel-Elements im Beitrag mit dem Titel „Wer will schon Dokumente ändern?“.

```
update delete input()/BTW2003/Beitrag/@status
       delete input()/BTW2003/Beitrag/Titel[. = "Wer will schon Dokumente ändern?"]/text()
```

Änderung 9: Löschen von Attributen und Textknoten

Die Ausführung dieser update-Anweisung auf Dokument 1 liefert:



Dokument 5: Änderung 9 angewandt auf Dokument

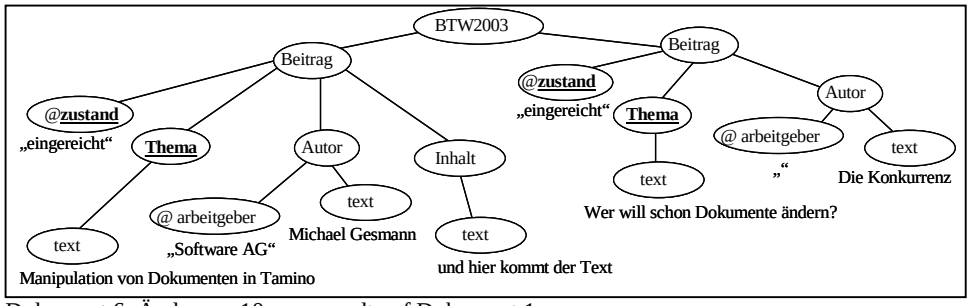
Rename

Für die Umbenennungsfunktion müssen alle Änderungsknoten Elemente, Attribute oder Knoten mit Verarbeitungsinstruktionen sein, die nach Ausführung der Funktion den angegebenen Namen haben.
Die folgende Anweisung nennt Elemente um, die über den Pfad BTW2003/Beitrag/Titel erreichbar sind, und macht Thema-Elemente daraus. Alle status-Attribute im Dokument werden in „zustand“ umbenannt.

```
update rename input()/BTW2003/Beitrag/Titel as"Thema"
       rename input()//@status as "zustand"
```

Änderung 10: Umbenennung von Elementen und Attributen

Auf Dokument 1 angewendet ergibt diese Operation folgenden Dokumentinhalt:



Dokument 6: Änderung 10 angewandt auf Dokument 1

Replace

Mit der Ersetzungsfunktion werden Änderungsknoten durch die neuen Werteknoten ersetzt. Im folgenden Beispiel wird das komplette Titel-Element durch ein "englisches" Title-Element ersetzt und der Titel selbst ebenfalls geändert:

```
update replace input()/BTW2003/Beitrag/Titel[.="Manipulation von Dokumenten in Tamino"]
            with <Title>Manipulation von Dokumenten in DBMS</Title>
```

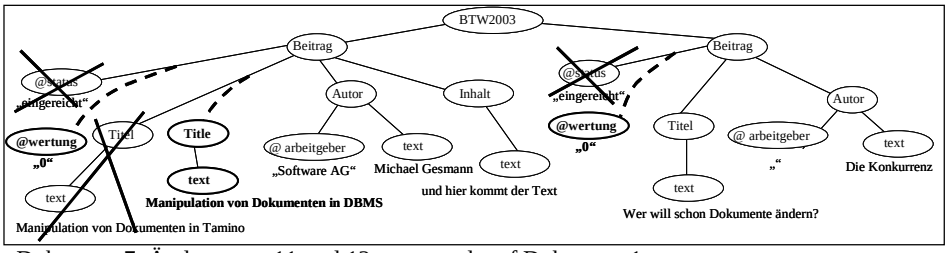
Änderung 11: Ersetzen von Elementen

Das status-Attribut wird durch ein wertung-Attribut mit dem Wert 0 ersetzt.

```
update replace input()/BTW2003/Beitrag/@status with attribute wertung{"0"}
```

Änderung 12: Ersetzen von Attributen

Wenn diese beiden Anweisungen nacheinander auf Dokument 1 angewandt werden, ergibt sich folgendes Ergebnis:



Dokument 7: Änderungen 11 und 12 angewandt auf Dokument 1

3.2 FLWUExpr

Die bisher aufgeführten Beispiele zeichnen sich dadurch aus, dass sowohl die Änderungs- als auch die Werteknoten durch einfache XPath-Ausdrücke bestimmt werden können. Diese XPath-Ausdrücke enthalten einfache Prädikate und liefern immer Knoten des Eingabedokuments. Genauso wie XPath-Anfragen auch durch XQuery-Sprachkonstrukte formuliert werden können, ist es für Änderungsoperationen möglich, mit For-Let-Where-Konstrukten zu arbeiten.

Die Titeländerung aus Abschnitt 2 läßt sich damit auch so formulieren:

```
update for $beitrag in input()/BTW2003/Beitrag
        where $beitrag/Autor = "Michael Gesmann" do
        replace $beitrag/Titel/text() with "Manipulation von Dokumenten in DBMS"
```

Änderung 13: Formulierung der Beispieländerung mit FLWUExpr

In vielen Fällen ist die Frage, ob ein Dokument mit FLWUExpr oder direkt mit den Änderungsoperationen formuliert wird, reine Geschmackssache. Darüber hinaus bieten FLWUExpr allerdings einen erweiterten Funktionsumfang. Mit den Variablenbindungen der For- und Let-Klauseln lassen sich Bedingungen formulieren, die über das hinausgehen, was XPath-Ausdrücke anbieten. Den Wert des in Änderung 12 eingefügten wertung-Attributs um 1 zu erhöhen, ist nur mit der folgenden Anweisung möglich:

```
update for $wertung in input()/BTW2003/Beitrag/@wertung do
            replace $wertung with attribute wertung {$wertung + 1}
```

Änderung 14: relative Wertveränderung von Attributen

3.3 Komplexere Szenarien

Alle bisherigen Beispiele zeichnen sich dadurch aus, daß jedem Änderungsknoten eindeutig eine einzige Änderung zugeordnet werden kann. Dies galt auch dann, wenn mehrere Elementaroperationen in einer XQuery angegeben waren. Wie Tamino Situationen behandelt, in denen dies nicht der Fall ist, wird im folgenden erläutert.

Die meisten in Abschnitt 2 vorgestellten Ansätze definieren, dass Elementaroperationen auch in der Reihenfolge ausgeführt werden sollen, in der sie angegeben werden. Dies ist hilfreich und sinnvoll, wenn Änderungen von Benutzern auf einzelnen Dokumenten durchgeführt wurden und anschließend analog in einer Datenbank nachvollzogen werden sollen. Deskriptiv beschriebenen Änderungen, die auf größeren Dokumenten oder Dokumentmengen ausgeführt werden sollen, nimmt dieser Ansatz aber auch Optimierungsmöglichkeiten. In Tamino hat die Reihenfolge, in der einzelne Änderungsoperationen angegeben werden, deshalb keine Bedeutung.

Aufgrund der Ausdrucksmächtigkeit von XQuery kann es so aber geschehen, dass ein Knoten sowohl als Änderungs- als auch als Werteknoten ausgewählt wird. Damit eindeutige Ergebnisse erzeugt werden können, wird festgelegt, dass die durchzuführenden Änderungen keinerlei Einfluß auf die Menge der Änderungsknoten und deren Inhalte haben sollen. Das gleiche gilt für die Werteknoten. Die neuen Werte werden ausschließlich auf den Ausgangsdaten berechnet, ohne von anderen Änderungen beeinflußt zu werden. Werteknoten entsprechen deshalb niemals Referenzen auf bereits existierende Knoten in Dokumenten sondern werden gegebenenfalls als neue Knoten kopiert.

Mit einer kleinen Auswahl weiterer Szenarien zeigt dieser Abschnitt, wie auch komplexere Aufgabenstellungen mit der präsentierten Spracherweiterung gelöst werden können.

Umstrukturierung von Dokumenten

Zunächst erhält jeder Beitrag eine laufende Nummer als Attribut. Weiterhin werden Titel und Autor-Elemente in einem Info-Element zusammengefaßt, wobei das Titel-Element gleichzeitig in Thema-Element umgenannt wird. Hier sind die Autor-Elemente und die Inhalte der Title-Elemente sowohl Änderungsknoten (als zu löschende Knoten) als auch Werteknoten (als einzufügende Knoten). Zunächst werden die Änderungs- und Werteknoten berechnet, ehe anschließend die eigentlichen Änderungen ausgeführt werden. So können Autor-Elemente und die Inhalte aus Titel-Elementen im neuen Info-Element eingesetzt werden.

Alle Operationen zusammen erfolgen mit Änderung 15 und liefern Dokument 8:

```
update for $beitrag in input()/BTW2003/Beitrag
        let $autor = $beitrag/Autor do (
insert attribute lfdNr="{$beitrag/position()}" into $beitrag
delete $beitrag/Titel
delete $autor
insert <Info>{$autor}<Thema>{$beitrag/Titel/text()}</Thema></Info> preceding $beitrag/*[1])
```

Änderung 15: Umstrukturierung eines Dokuments

```
<?xml version="1.0"?>
<BTW2003>
   <Beitrag lfdNr="1" status="eingereicht">
     <Info> <Autor arbeitgeber="Software AG">Michael Gesmann</Autor>
              <Thema>Manipulation von Dokumenten in Tamino</Thema></Info>
     <Inhalt> und hier kommt der Text </Inhalt></Beitrag>
   <Beitrag lfdNr="2" status="eingereicht">
      <Info> <Autor arbeitgeber="">Die Konkurrenz</Autor>
               <Thema>Wer will schon Dokumente ändern?</Thema></Info></Beitrag>
</BTW2003>
```

Dokument 8: Änderung 15 angewandt auf Dokument 1

Zusammenfügen von Dokumenten

Nachdem die Beiträge zur Begutachtung verteilt wurden, treffen Gutachten ein, die zunächst in einer separaten Datenquelle abgelegt werden. Das Dokument mit den Gutachten sieht für die folgenden Beispiele folgendermassen aus:

```
<?xml version="1.0"?>
<BTW2003Gutachten>
   <Gutachten beitragNr="1" urteil="ausgezeichnet">
       <Gutachter arbeitgeber="Software AG">Manfred Michels</Gutachter>
       <Details> Jetzt weiß ich endlich was wir da machen </Details>    </Gutachten >
   <Gutachten beitragNr="1" urteil ="bescheiden">
      <Gutachter arbeitgeber="">Die Konkurrenz</Gutachter>
      <Details>Wer will schon Dokumente ändern?</Details>    </Gutachten >
</BTW2003Gutachten>
```

Dokument 9: Eingabedokument mit Gutachten

Die Gutachten aus Dokument 9 können folgendermaßen an das Ende ihrer zugehörigen Beiträge in Dokument 8 kopiert werden:

```
update for $beitrag in input()/BTW2003/Beitrag
    let $gutachten := input()/BTW2003Gutachten/Gutachten [@beitragNr = $beitrag/@lfdNr] do
        insert $gutachten into $beitrag
```

Änderung 16: Kopieren von Knoten in ein anderes Dokument

Das BTW2003-Dokument sieht danach folgendermaßen aus:

```
<?xml version="1.0"?>
<BTW2003>
   <Beitrag lfdNr="1" status="eingereicht">
     <Info> <Autor arbeitgeber="Software AG">Michael Gesmann</Autor>
            <Thema>Manipulation von Dokumenten in Tamino</Thema>    </Info>
                  <Inhalt> und hier kommt der Text </Inhalt>
     <Gutachten beitragNr="1" urteil="ausgezeichnet">
        <Gutachter arbeitgeber="Software AG">Manfred Michels</Gutachter>
        <Details> Jetzt weiß ich endlich was wir da machen </Details>    </Gutachten >
     <Gutachten beitragNr="1" urteil ="bescheiden">
        <Gutachter arbeitgeber="">Die Konkurrenz</Gutachter>
        <Details>Wer will schon Dokumente ändern?</Details>    </Gutachten >
   </Beitrag>
   <Beitrag lfdNr="2" status="eingereicht">
     <Info> <Autor arbeitgeber="">Die Konkurrenz</Autor>
            <Thema>Wer will schon Dokumente ändern?</Thema>  </Info>    </Beitrag>
</BTW2003>
```

Dokument 10: Änderung 16 angewandt auf die Dokumente 8 und 9

Herausschneiden von Dokumentteilen

Abschließend werden aus Dokument 10 alle Beiträge gelöscht, zu denen kein Gutachter etwas liefern wollte/konnte. Außerdem werden alle Gutachten gelöscht, bei denen der Gutachter den gleichen Arbeitgeber hat wie einer der Autoren:

```
update for $beitrag in input()/BTW2003/Beitrag[empty(Gutachten)]
       for $gutachten in input()BTW2003/Beitrag/Gutachten
                                    [Gutachter/@arbeitgeber = ../Info/Autor/@arbeitgeber]
do ( delete $beitrag
     delete $gutachten )
```

Änderung 17: löschen mehrere Elemente

Und hier nun das Ergebnis all der Arbeit:

```
<?xml version="1.0"?>
<BTW2003>
<Beitrag lfdNr="1" status="eingereicht">
     <Info> <Autor arbeitgeber="Software AG">Michael Gesmann</Autor>
            <Thema>Manipulation von Dokumenten in Tamino</Thema>    </Info>
                  <Inhalt> und hier kommt der Text </Inhalt>
     <Gutachten beitragNr="1" urteil ="bescheiden">
        <Gutachter arbeitgeber="">Die Konkurrenz</Gutachter>
        <Details>Wer will schon Dokumente ändern?</Details>    </Gutachten >
   </Beitrag>
</BTW2003>
```

Dokument 11: Änderung 17 angewandt auf Dokument 10

3.4 Zusammenfassung

Dieser Abschnitt hat dargestellt, wie elementare Änderungsoperationen mit einfachen Spracherweiterungen in XQuery angeboten werden können. Zusammen mit der in XQuery selber vorhandenen Funktionalität zur Suche von Dokumenten und Dokumentinhalten sowie Attribut- und Elementkonstruktoren genügen die gezeigten Operationen, um die in Abschnitt 2 aufgezählte Basisfunktionalität zu realisieren. Anhand von zunächst einfachen und später auch komplexeren Beispielen wurden die Operationen erläutert. Die Reihenfolge, in der Änderungsoperationen angegeben werden, hat keinen Einfluß auf deren Ausführungsreihenfolge. Als Ausführungsmodell werden Änderungs- und Werteknoten berechnet, bevor die eigentlichen Veränderungen an den Dokumenten vorgenommen werden.

4 Angabe mehrerer Änderungen auf einem Knoten

Die im vorangegangenen Abschnitt beschriebene Unabhängigkeit der Ausführungsreihenfolge kann zu Problemen führen, wenn das Ergebnis nicht mehr eindeutig charakterisiert wird. Was darunter zu verstehen ist, und wie Tamino damit umgeht, wird in diesem Abschnitt ausgeführt.

4.1 Konflikte

Eine komplex formulierte Änderungsoperation setzt sich aus den im vorangegangenen Abschnitt erklärten Elementaroperationen "Insert", "Delete", "Rename" und "Replace" zusammen. Werteknoten sind bei Insert und Replace-Operationen immer eindeutig festgelegt. Dagegen können auf einem Änderungsknoten mehrere Elementaroperationen ausgeführt werden. In einigen Fällen ist dies mit dem oben beschriebenen Ausführungsmodell problemlos möglich, zum Beispiel wenn ein Element umbenannt wird und gleichzeitig ein neues Attribut erhält. Es können aber auch Probleme auftreten, zum Beispiel, wenn zwei Umbenennungen auf einem Knoten ausgeführt werden sollen. Aufgrund der unspezifischen Ausführungsreihenfolge steht in diesem Fall nicht mehr fest, wie das Endergebnis aussieht. Je nach interner Ausführungsreihenfolge ergäben sich unterschiedliche Ergebnisdokumente. Solche Problemfälle werden Konflikte genannt.

Ein **Konflikt** tritt dann auf, wenn

1. eine Elementaroperation ausgeführt werden soll, die aber aufgrund anderer Elementaroperationen keine Auswirkung auf das Ergebnis hat, oder wenn
2. das Ergebnis der Operationen zu einem nicht wohlgeformten Dokument führt, oder wenn
3. das Ergebnis der an einem Änderungsknoten auszuführenden Elementaroperationen nicht eindeutig ist.

Eine Erläuterung der möglichen Konflikte erfolgt zunächst für den Fall, daß der Änderungsknoten ein Element ist.

Fall 1: unsinnige Operationen
Der erste Fall tritt immer dann ein, wenn ein Knoten mit seinen Nachfolgern gelöscht oder ausgewechselt wird, gleichzeitig aber weitere Änderungsoperationen auf den aus dem Dokument entfernten Knoten definiert werden. Somit wurde mindestens eine Änderungsoperation angegeben, die als überflüssig oder unsinnig angesehen wird. Als Spezialfall stellt sich diese Situation auch immer dann ein, wenn auf einem Änderungsknoten mehr als nur eine Lösch- oder Ersetzungsoperation definiert wird.

Fall 2: nicht wohlgeformte Dokumente
Der zweite Fall tritt dann ein, wenn zwei Attribute gleichen Namens eingefügt werden sollen oder zu einem bereits existierenden Attribut ein weiteres Attribut mit gleichem Namen hinzugefügt wird. Das Einfügen von Attributen darf allerdings temporär Dokumente erzeugen, die nicht wohlgeformt sind. So ist es möglich, ein Attribut neu einzufügen, obwohl es im Element schon ein Attribut gleichen Namens gibt, wenn das bereits existierende Attribut auch gelöscht wird. Diese Eigenschaft trägt dem Gedanken Rechnung, dass keine Ausführungsreihenfolge angegeben wird. Außerdem geht sie konform damit, dass auch die Dokumentvalidierung nicht nach jeder einzelnen Änderung erfolgt, sondern das Dokument erst nach Ausführung aller Elementaroperationen wieder gültig sein muß.

Fall 3: nicht kommutative Operationen
Der dritte Fall tritt dann ein, wenn zwei Elementaroperationen auf einem Änderungsknoten nicht kommutativ sind, das heißt, das Ergebnis der Elementaroperationen von der Reihenfolge ihrer Ausführung abhängt. Zum einen ist dies der Fall bei zwei „Rename“-Operationen mit unterschiedlichen Namen, für die nicht klar ist, welchen Namen der zu ändernde Knoten am Ende hat. Ausserdem tritt diese Situation bei zwei „Insert ... into ...“-Operationen auf einem Änderungsknoten auf, die keine Attribute einfügen. Wenn jede „Insert ... into ...“-Operation einen Knoten neu einfügt, ist am Ende nicht klar, in welcher Reihenfolge diese Knoten im Ergebnis erscheinen werden. Würde nur ein „Insert ... into ...“ mit einer Sequenz der beiden Knoten angegeben, wäre die Reihenfolge eindeutig.
Was für das „Insert ... into ...“ gilt, trifft in ähnlicher Form auch für das „Insert ... preceding ...“ und „Insert ... following...“ zu. Die Reihenfolge der neu eingefügten Knoten im Änderungsknoten ist nicht festgelegt, wenn es mehrere solcher Operationen an der gleichen Stelle in der Kindliste des Änderungsknoten gibt.
Hier beachte man, dass das positionsbezogene Einfügen auf dem Elternknoten und nicht auf den in der Operation angegebenen Geschwisterknoten als Änderungsknoten definiert ist. Dadurch liefert ein solches Insert auch dann noch ein definiertes Ergebnis, wenn der Geschwisterknoten(!) aus dem Dokument gelöscht wird. Zwei „Insert ... preceding ...“ oder „Insert ... following ...“ Operationen führen nur dann zu einem Konflikt, wenn sie sich auf die gleiche Position in der Kindliste des Änderungsknotens beziehen, ansonsten können die Operationen problemfrei ausgeführt werden.

Zusammenfassung
Tabelle 2 faßt zusammen, wann Konflikte dieser Art bei Elementaroperationen an **einem** Änderungsknoten auftreten. Die Insert-Operation wird in die vier Spezialfällen unterschieden: "InsertAttr" fügt ein neues Attribut ein und ist nur bei Elementknoten möglich. "InsertInto" deckt alle übrigen Fälle der „insert ... into ...“-Anweisung ab. „InsertPrec” und „InsertFoll“ stehen für die beiden verbleibenden Formen der Insert-Anweisung. Ein Haken beschreibt den Fall, dass beide Operationen in beliebiger Reihenfolge ausgeführt werden können, ohne dass das Endergebnis davon betroffen ist. Die ① beschreibt einen Konflikt nach Fall 1, die ② einen Konflikt nach Fall 2 und die ③ einen Konflikt durch nicht kommutative Operationen. Die ④ charakterisiert Konflikte durch Operationen, die an der gleichen Stelle innerhalb der Kindliste eines Änderungsknotens ausgeführt werden sollen.

Operationen auf einem Knoten	*Delete*	*Replace*	*Rename*	*Insert Attr*	*Insert Into*	*Insert Prec*	*Insert Foll*
Delete	①	①	①	①	①	①	①
Replace		①	①	①	①	①	①
Rename			③	✓	①	✓	✓
InsertAttr				②	✓	✓	✓
InsertInto					③	✓	✓
InsertPrec						④	✓
InsertFoll							④

Tabelle 2: Konflikte bei mehreren Operationen auf einem Knoten

Weitere Aspekte
Für Änderungsoperationen auf Attributen und „Processing Instructions“ gilt ebenfalls die obige Tabelle, wobei nur die Operationen Delete, Replace und Rename möglich sind. Für alle weiteren Knoten gilt die Tabelle mit der Einschränkung auf Delete und Replace.

Wie bei Elementen kommen auch bei anderen Knoten nach Fall 1 Konflikte hinzu, wenn auf diesen Knoten ein Delete oder ein Replace definiert ist und auf irgendeinem seiner Nachfolger weitere Operationen definiert werden. Diese weiteren Operationen fallen dann in die Kategorie ①.

Konflikte sind auf Dokumenten definiert, und ob ein Konflikt tatsächlich auftritt oder nicht, hängt stark von den Dokumenten ab, auf die eine Änderungsoperation angewandt wird. Deshalb ist eine statische Bestimmung solcher Konflikte, zum Beispiel durch einen Compiler, nicht möglich. Statt dessen muss die Konflikterkennung dynamisch zur Ausführungszeit einer Änderungsoperation erfolgen. In der Konsequenz kann das Einfügen eines neuen Dokumentes dazu führen, dass eine Änderungsoperation auf einen Konflikt stößt, oder umgekehrt kann das Löschen eines Dokuments dazu führen, dass eine Operation konfliktfrei ausgeführt werden kann.

4.2 Beispiel und Konfliktbehandlung

Anhand eines einfachen Beispiels sei ein Konflikt erläutert. Dokument 10 diene hier als Eingabe und es werde die Änderung 18 ausgeführt.

```
<?xml version="1.0"?>
<BTW2003>
    <Beitrag lfdNr="1" status="eingereicht">
        <Autor arbeitgeber="Software AG">Michael Gesmann</Autor>
        <Thema>Manipulation von Dokumenten in Tamino</Thema>
        <Inhalt> und hier kommt der Text </Inhalt>
       <Gutachten urteil="abgelehnt">
           <Gutachter>Konkurrenz</Gutachter>
       </Gutachten>
    </Beitrag>
    <Beitrag lfdNr="2" status="eingereicht">
        <Autor arbeitgeber="">Die Konkurrenz</Autor>
        <Thema>Wer will schon Dokumente ändern?</Thema>
        <Gutachten urteil="unentschieden">
           <Gutachter></Gutachter>
        </Gutachten>
    </Beitrag>
</BTW2003>
```

Dokument 10: Eingabedokument für folgende Beispiele

```
update for $beitrag in input()/BTW2003/Beitrag[Gutachten] do
        ( replace $beitrag/@status with attribute status {"beurteilt"}
          delete $beitrag/Gutachten[Gutachter=""]
          rename$beitrag//@urteil as bewertung )
```

Änderung 18: Beispiel, mit dem Konflikte erzeugt werden

Für den ersten Beitrag bekommt das status-Attribut einen neuen Wert, es wird kein Knoten gelöscht und ein urteil-Attribut umgenannt. Für den zweiten Beitrag wird ebenfalls der Wert des status-Attributs verändert: es soll das darin enthaltene Gutachten gelöscht werden, und das in dem Gutachten enthaltene urteil-Attribut soll umbenannt werden. Diese letzte Umbenennung ist unnötig, da das umgebende Element sogar gelöscht werden soll. Diese Situation führt zu einem Konflikt.

Konflikterkennung auf der Basis der oben vorgegebenen Definition für Konflikte zu betreiben, ist ein sehr strikter Ansatz, der davon ausgeht, dass alle Änderungsoperationen so eindeutig abgefaßt werden, dass letztendlich keine Widersprüche oder Ungenauigkeiten zur Ausführungszeit mehr auftreten. Wenn solche Konflikte auftreten, dann hat die Anwendung anscheinend noch nicht vollständig die Reichweite der Operation erfaßt. In diesen Fällen lehnt Tamino die Änderungsoperation ab.

5 Zusammenfassung und Ausblick

Dieser Beitrag ist der Frage nachgegangen, wie Änderungen in XML-Dokumenten formuliert werden können, damit sie in einer Datenbank ausgeführt werden können. Zunächst wurden bestehende Ansätze beschrieben und bewertet. Als ausdrucksstärkster Ansatz wurden Erweiterungen von XQuery angesehen. Dieser Ansatz wurde für eine Realisierung im XML-DBMS Tamino der Software AG ausgewählt. Mit den dort implementierten Erweiterungen von XQuery entsteht eine Sprache, die sowohl die Änderungsoperationen selber, die Suche nach zu verändernden Knoten als auch die Bestimmung der neuen Werte einheitlich beschreiben kann. Eine Änderungsoperation kann sich aus mehreren Elementaroperationen zusammensetzen. Die Reihenfolge, in der die Elementaroperationen angegeben werden, hat in Tamino keinen Einfluß auf das Ergebnis. Die Spracherweiterungen und deren Semantik wurde anhand zahlreicher Beispiele dargestellt. Der Beitrag zeigt, wie diese Forderung nach Reihenfolge-unabhängigkeit zu Konflikten führen kann. Konfliktbehaftete Änderungsoperationen werden mit einer Fehlermeldung abgelehnt.
Die praktische Erprobung der Konzepte durch Anwendungen steht noch aus. Insbesondere die Behandlung von Konflikten muss dort ihre Praxistauglichkeit zeigen. Ferner läßt die vorgestellte Realisierung eine Reihe offener Fragen, die weiter untersucht werden müssen:

- [Le01] hat in seiner Spracherweiterung mit einem if-then-else-Konstrukt auch die bedingte Ausführung von Änderungsoperationen vorgesehen. Dieses wurde bisher nicht berücksichtigt.
- Mit dem Dokument-Konstruktor von XQuery können auch neue Dokumente eingefügt werden. Das Löschen eines Dokumentknotens führt zum Löschen des ganzen Dokuments. Es ist noch unklar, inwieweit sich diese Erweiterungen eignen, Systemschnittstellen für das Einfügen und Löschen von Dokumenten zu ersetzen.
- Die Konflikterkennung wie sie in Abschnitt 4 beschrieben wurde, ist sehr restriktiv. Es könnten mehr Freiheiten zugelassen werden, indem mehr Anwendungssemantik ausgenutzt wird. So muß es nicht unbedingt ein Fehler sein, wenn Änderungen auf Knoten definiert sind, die auch gelöscht werden. Es ist genauso gut denkbar, den Netto-Effekt, in diesem Fall das Löschen, zu berechnen und nur diese Änderung durchzuführen. Zur Zeit geht Tamino den restriktiven Weg in der Annahme, dass zukünftigen Versionen an dieser Stelle im oben genannten Sinne geöffnet werden können. Hingegen ist es fast unmöglich, einmal angebotene Funktionalität wieder herauszunehmen. Hier muß auch abgewartet werden, was spätere W3C Empfehlungen festlegen werden.
- In manchen Anwendungen spielt die Reihenfolge, in der Änderungen ausgeführt werden sollen, eine wesentliche Rolle. Ein Beispiel hierfür hat Abschnitt 3.3 gezeigt. Dort wurden zuerst Gutachten zu den BTW-Beiträgen kopiert, ehe anschließend die Beiträge gelöscht wurden, zu denen es keine Gutachten gab. Eine solche Ausführungsreihenfolge kann mit der präsentierten Spracherweiterung zur Zeit nicht in einer Anweisung formuliert werden. Statt dessen müssen entweder sehr komplexe Prädikate für die einzelnen Operationen definiert werden, oder die einzelnen Änderungsoperationen können nur als individuelle Update-Anweisungen unabhängig voneinander in der gewünschten Reihenfolge an das System gerichtet werden. Diese Forderung nach einer benutzerdefinierten Reihenfolge der Operationen steht zunächst

im Widerspruch zur systembestimmten Ausführung der Elementaroperationen. Wie die Sprache erweitert werden kann, so dass auch eine Sequenz von Operationen in einer einzigen Anweisung ausgeführt werden kann, ist bisher nicht klar.

- Aus Platzgründen läßt dieser Beitrag alle Fragen hinsichtlich der Implementierung der aufgeführten Schnittstelle offen. Hier sei lediglich auf zwei prominente Problembereiche hingewiesen. Es ist keine triviale Aufgabe, Dokumente nur partiell zu validieren, wenn Konsistenzbedingungen wie zum Beispiel Eindeutigkeit von Werten in gewissen Dokumentteilen oder Knoten vom Typ ID bzw. IDREF berücksichtigt werden müssen. Ähnliches gilt je nach gewählter Indeximplementierung auch für partielle Änderungen in Indexen.

Literaturverzeichnis

[Db2] IBM DB2 Universal Database, XML Extender - Administration and Programming, Version 7, ftp://ftp.software.ibm.com/ps/products/db2/info/vr7/pdf/letter/db2sxe70.pdf

[Le01] Lehti, Patrick: "Design and Implementation of a Data Manipulation Processor for an XML Query Language", August 2001, Diplomarbeit an der Technischen Universität Darmstadt - FB Elektrotechnik und Informationstechnik

[Ora9i2] Oracle9i Database Online Documentation Release 2, http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/index.htm

[QuiP00] Quip Version 2.2.1, Software AG's XQuery prototype, http://developer.softwareag.com/tamino/quip

[SiXDML02] "Simple XML Data Manipulation Language Draft", Working Draft 2002-06-23, http://www.xmldb.org/sixdml/sixdml-lang.html

[SQLServ2000] Using UpdateGrams to Modify Data, SQLXML 3.0, http://msdn.microsoft.com/library/default.asp?url=/library/enus/sqlxml3/htm/updategram_5kkh.asp

[Tam41] Dokumentation zu Tamino Version 4.1, (noch nicht erschienen)

[TIHW01] Tatarinov, Ives; Halevy, Weld: "Updating XML", Proceedings ACM SIGMOD 2001, pp. 413-424, May21-24, Santa Barbara, California, USA

[XIS312] eXcelon Product Information, Extensible Information Server Version 3.1.2, http://support.exceloncorp.com/doc/full/xml/xis312/webhelp/m_upwiz.htm

[XNs99] Namespaces in XML, Empfehlung des W3C, http://www.w3.org/TR/REC-xml-names

[XP99] XML Path Language (XPath), Empfehlung des W3C, http://www.w3.org/TR/xpath

[XQ02] XML Query, Arbeitsmaterial der “XML Query Working Group” im W3C, http://www.w3.org/XML/Query

[XSD01] XML Schema Empfehlung des W3C, http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1, http://www.w3.org/TR/xmlschema-2

[XUp02] "XUpdate - XML Update Language", Working Draft 2000-09-14, http://www.xmldb.org/xupdate/xupdate-wd.html

[Xup02R] "Requirements for XML Update Language", Working Draft der XML:DB Initiative 2000-11-24, http://www.xmldb.org/xupdate/xupdate-req.htm

XML in der Oracle Datenbank „relational and beyond“

Ulrike Schwinn
Oracle Deutschland GmbH
Riesstrasse 25
D-80992 München
Ulrike.Schwinn@oracle.com

Abstract: In Geschäfts- und B2B-Anwendungen wird XML zunehmend als das Format für elektronisches Publizieren und den Austausch von Dokumenten eingesetzt. Richtiges Speichern von XML-Dokumenten ist dabei ein viel diskutiertes Thema. So werden beispielsweise beim nicht nativen Speichern Webinhalte und Applikationsdaten entweder in relationalen Tabellen, im Dateisystem oder in beidem gespeichert. Die besonderen Merkmale von XML können dabei jedoch nicht berücksichtigt werden. Bei großen Mengen von Austauschdaten stellt sich überhaupt die Frage, ob diese Speicherungsform XML-Applikationen überhaupt gerecht werden kann. Eine gute Lösung bieten hingegen die so genannten nativen XML-Datenbanken. In diesem Artikel wird am Beispiel des neuesten Oracle-Datenbank-Release das Konzept einer nativen XML-Datenbank vorgestellt.

1 Einleitung

Der Zunahme von XML-Applikationen (eXtended Markup Language) und die stärkere Nutzung von Webservices (1) erfordert, Werkzeuge zum effektiven Speichern und Verarbeiten von XML-Dokumenten.
Wählt man die Möglichkeit, XML-Dokumente in einer Datenbank zu speichern, kann dadurch sogar der Funktionsumfang der XML-Applikationen erweitert werden. Dazu tragen typische Eigenschaften der Datenbanken (DB) bei, wie zum Beispiel das Management von konkurrierenden Zugriffen, Sicherheit, Sperrverhalten, Datenintegrität mit Constraints und Trigger, sowie eine einfache zentralen Verwaltung oder ein ausgeklügeltes Backup- und Recovery-System.

Wenn aus Sicht der XML-Applikation, zum Beispiel einer Kataloganwendung oder Lastenheftverwaltung, die Entscheidung zur Speicherung in der Datenbank gefallen ist, bedeutet dies, dass die XML-Daten dauerhaft und zentral zur Verfügung gestellt werden sollen. Bei der Auswahl der Speicherung kommt es auf deren Eignung an- sowohl der Applikation als auch der Eigenschaft von XML-Daten an. So bietet sich beispielsweise das rein relationale DB-Modell an, um XML-Dokumente, daten- als auch dokumentzentrisch abzuspeichern. Allerdings stößt man schnell an die Grenzen des rein relationalen Ansatzes. Dieses Modell bildet nur eine kleine Untermenge von XML-Daten ab. XML-Charakteristiken, wie speicherspezifische Informationen etwa die Reihenfolge, dokumentspezifische Informationen oder Kommentar können dabei nicht berücksichtigt werden.

Objektrelationale Datenbanken dagegen erweiterte Möglichkeiten indem sie mit einer Abbildung auf Objekte. Das folgende Beispiel illustriert diese Möglichkeit:

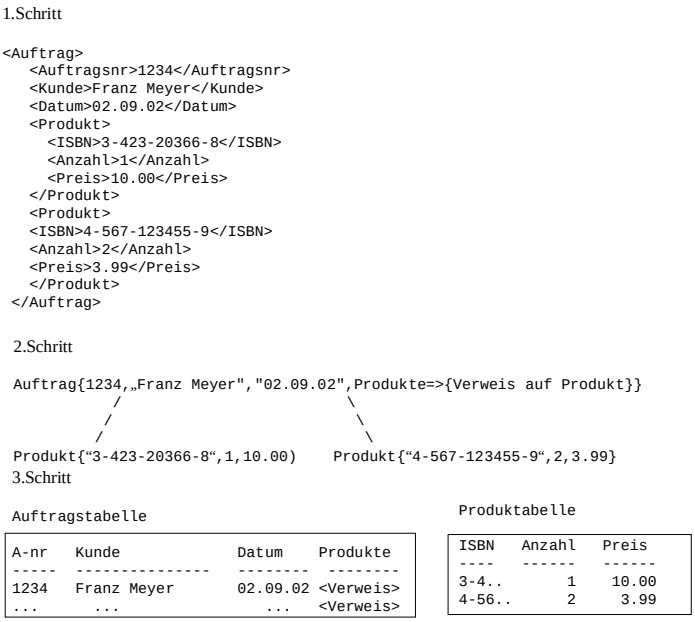


Abb. 1.1: Schrittweises Überführen eines XML-Dokuments in eine objektrelationale Speicherform

Über die reinen Speicherungsformen hinaus müssen die Verarbeitungsmöglichkeiten innerhalb der DB berücksichtigt werden. Das heißt, es müssen Zugriffsmethoden, Mechanismen zum Darstellen oder Wiederherstellen von XML-Dokumenten ergänzt beziehungsweise sichergestellt werden.
Unabhängig von den dargestellten Abbildungsformen sind daher die so genannten nativen XML-Datenbanken seit geraumer Zeit im Gespräch. Die Oracle-Datenbank ist ein Beispiel für solch eine Datenbank. Die Definition von Kimbro Staken (2) gibt vor, welche Eigenschaften eine Datenbank haben muss, um XML nativ abspeichern zu können: Native XML Datenbanken

- definieren ein logisches Datenmodell, nach dem sie die XML-Dokumente speichert und wiederherstellt. Beispiele für solche Modelle sind XPATH, XML-Schema, Modelle, die das „Document Object Model“ (DOM) unterstützen und andere,
- haben das XML-Dokument als eine logische Einheit wie Zeilen einer Tabellen in der Datenbank,
- sind unabhängig von den darunter liegenden Speicherungsformen.

2 So speichert Oracle XML

Trotz der Unabhängigkeit von der Speicherform aus Sicht der XML-Applikation und des Datenzugriffs, ist das Speicherdesign für die Performanz und Konkurrenzfähigkeit der Applikation relevant. Daher wird im Folgenden ein kurzer Überblick zu den unterschiedlichen Speicherformen innerhalb der Oracle-Datenbank gegeben.
Wie im oberen Abschnitt schon erwähnt, werden XML-Dokumente durch ihre Charakteristik in dokument- und datenzentrische Dokumente unterteilt. Die Speicherform als Character Large Object (CLOB) ermöglicht dokumentzentrisches Speichern. Die objektrelationale Speichervariante kommt dem datenzentrischen Dokument-Typ entgegen. Damit die Datenbank zusätzliche XML-Zugriffsweisen bereitstellt, steht der Datentyp XMLTYPE zur Verfügung. Bei einfacher Angabe von XMLTYPE wird dadurch eine dokumentzentrische Speicherung als CLOB möglich.

```
CREATE TABLE auftrag of XMLTYPE;
CREATE TABLE auftrag (auftrags_nr NUMBER,
                      A_Beschreibung XMLTYPE);
```

Abb. 2.1: Beispiele für die dokumentzentrische Speicherung

Um die objektrelationale Speicherung beeinflussen zu können, muss die Datenbank eine Beschreibung der XML-Strukturen erhalten. Dies geschieht durch den Einsatz von standardkonformen annotated XML-Schemas, die die Oracle-spezifischen Beschreibungen als Attribute in den Schema-Annotations mit liefert. Auf diese Weise können zum Beispiel die per Default vergebenen Systemnamen oder Speicherformen für Objekte durch sinnvolle Namen und speziellen Speicherformen ersetzt werden. Damit ein solches XML-Schema nicht proprietär wird, kommt die Technik der Namensräume (Namespace), ein Bestandteil des XML-Standards, zum Einsatz. Durch Angabe eines speziellen Oracle-Namespace werden diese Attribute eindeutig von W3C-Schema-Informationen abgegrenzt. Solche Schemas können problemlos von beliebigen - nicht Oracle spezifischen - Werkzeugen benutzt werden, da der Standard vorsieht, dass unbekannte Annotations einfach ignoriert werden.
Dies hat zur Folge, dass in solchen XML-Schemas, in der Regel mindestens zwei Namensräume angegeben werden – einen für W3C, http://www.w3c.org/2001/XMLSchema und einen für die Oracle-Charakteristiken, http://xmlns.oracle.com/xdb.
Folgende Abbildung zeigt die objektrelationale Speicherung von Abbildung 1.1 in der Datenbank. Die Oracle-spezifischen Annotations sind mit dem Aliasnamen „xdb“ gekennzeichnet.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xdb="http://xmlns.oracle.com/xdb"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.oracle.com/xsd/auftrag.xsd"
      elementFormDefault="qualified"
     attributeFormDefault="unqualified">
  <xs:element name="Auftrag" xdb:SQLName="AUFTRAG"
     xdb:SQLType="AUFTRAG_T"xdb:defaultTable="AUFTRAG_TAB">
     <xs:complexType>
       <xs:sequence>
        <xs:element name="Auftragsnr"
            type="xs:integer" xdb:SQLName="A_NR"
            xdb:SQLType="INTEGER" />
         <xs:element name="Kunde" type="xs:string"
             xdb:SQLName="KUNDE"
              xdb:SQLType="VARCHAR2" />
         <xs:element name="Datum" type="xs:date"
          xdb:SQLName="DATUM" xdb:SQLType="DATE" />
         <xs:element name="Produkt"
             maxOccurs="unbounded"
             xdb:SQLName="PRODUKTE"
             xdb:SQLType="PRODUKT_T"
             xdb:defaultTable="PRODUKT_TAB"
             xdb:SQLInline="false">
          <xs:complexType>
            <xs:sequence>
                 <xs:element name="ISBN"
                .....
            </xs:sequence>
          </xs:complexType>
             ....
  </xs:schema>
```

Abb. 2.2: Schema auftrag.xsd für das XML-Dokument auftrag.xml

Folgende Annotations sind im Schema auftrag.xsd enthalten:

- SQLName für den Objektnamen,
- SQLType für den Oracle Datenbanktyp,
- SQLInline zur Festlegung, ob ein XML-Dokument in einem oder mehreren Objekten gespeichert wird,
- DefaultTable für den Tabellennamen.

Darüber hinaus können mit diesen Annotations aber auch zusätzliche Constraints, Partitionierungen oder auch Speicherung von XML-Fragmenten in CLOBs (Character Large Objects), in speziellen Feldern und Arrays, in Oracle-Terminlogie auch Varrays oder Nested Tables genannt, mit diesen Annotations definiert werden.

Nach Registrieren des Schemas von Abbildung 2.2, sind die Objekttypen AUFTRAG_T und PRODUKT_T und die zugehörigen Objekttabellen und Arrays durch die Annotations in der Datenbank angelegt worden.
Nachfolgende schemakonforme XML-Dokumente wie z.B. auftrag.xml werden dann in den entsprechenden Objekttabellen wie PRODUKT_TAB und AUFTRAG_TAB

gespeichert. Zur Identifikation des zugehörigen XML-Schemas dient dabei eine eindeutige URL, die beim Registrieren des Schemas festgelegt wurde. Diese URL, wird wie der Standard es vorgibt, im Attribut `xsi:schemaLocation` im XML-Dokument mitgegeben - das erste Argument liefert den zugehörigen Namensraum und das Zweite die eindeutige URL, die beim Registrieren festgelegt wurde.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Auftrag xmlns=http://www.oracle.com/xsd/auftrag.xsd
         xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xsi:schemaLocation="http://www.oracle.com/xsd/auftrag.xsd
                     http://www.oracle.com/xsd/auftrag.xsd">
  <Auftragsnr>...
```

Abb. 2.3: XML-Dokument zum Schema auftrag.xsd

Als dritte Variante lassen sich auch semistrukturierte Dokumente, die weder rein Daten- noch Dokument-zentrisch sind, als Form der objektrelationalen Speicherung realisieren. Möglich wird das durch die Technik, Teilfragmente als CLOB im XML-Schema anzugeben. Ein Beispiel für solch eine Anwendung wäre ein Presseartikel, der teilweise aus einem stark strukturiertem Dokument-Header (Autor, Rubrik, Datum) und einem unstrukturierten Dokument-Body (Text) besteht.

Die Abbildung zeigt einen Ausschnitt aus solch einem Schema:

```
<xs:schema xmlns:xdb=http://xmlns.oracle.com/xdb
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
  <xs:element name="ARTIKEL"
      xdb:defaultTable="ARTIKEL_TAB"
      xdb:tableProps="partition by range
      (xmldata.head.DAT) (partition p1 values less than
      (to_date('1998-12-31','YYYY-MM-DD')),... partition
      p6 values less than (to_date('2003-12-31','YYYY-MM-
      DD')))">
    <xs:complexType xdb:SQLType="ARTIKEL_T">
      <xs:sequence>
         <xs:element name="HEAD"
          type="HEADType" xdb:SQLName="HEAD" />
          <xs:element name="BODY" type="BODYType"
            xdb:SQLName="BODY" xdb:SQLType="CLOB"
            />
        </xs:sequence> ...
```

Abb. 2.4: Beispiel einer semistrukturierten Speicherung

3 Der Funktionsvorrat im Überblick

Neben den verschiedenen Speicherkonzepten bietet die Oracle-Datenbank wichtige Erweiterungen zur Unterstützung von XML-Applikationen.

In einer objektrelationalen Datenbank ist es natürlich möglich, über SQL- Zugriffe Daten abzufragen oder zu verändern. Dabei ist jedoch die Integration der XPATH-Funktionalität (XML Path Language) in den SQL-Abfragen wichtig, um XML-Anfragen standardgemäß implementieren zu können. Um dabei nicht jedes Mal den ressourcenintensiven DOM- Baum aufbauen zu müssen, besteht bei objektrelationaler Speicherung die Möglichkeit, den Zugriff direkt auf die Objekte abzubilden. Diese Methode, auch Query Rewrite genannt, beschleunigt nicht nur die Abfrage, sondern verkleinert auch den Speicherverbrauch. Das folgende Beispiel illustriert solch ein Query Rewrite.

Folgenden Anfrage

```
SELECT EXTRACT (value(a), '/Auftrag/Kunde') FROM auftrag_tab a
```

wird automatisch umgeschrieben auf

```
SELECT a.xmldata.kunde FROM auftrag_tab a
```

Darüber hinaus bietet die Datenbank die Möglichkeit, verschiedene Indizierungsmechanismen einzusetzen, um die Abfragen zu beschleunigen. Der B*-Index zum Beispiel beschleunigt Abfragen von Daten, die objektrelational gespeichert sind. Der Volltextindex, der besonders bei dokumentzentrischen Daten wichtig ist, unterstützt die XPATH-Syntax und kann unabhängig von der Speicherung zur Beschleunigung der Volltextsuche eingesetzt werden.

Nicht zuletzt können ändernde Zugriffe, die konkurrierend stattfinden sollen, entweder durch Änderung und Austausch des gesamten Dokuments oder aber durch Bearbeiten nur eines Teilbaumes erfolgen. Um konkurrierenden Zugriffe auf Teilbäume ohne Sperren des gesamten Dokuments zu ermöglichen, ist speziell für die objektrelationale Speicherung die SQL-Funktion XMLUPDATE hinzugefügt worden. Sie ermöglicht den so genannten „piecewise Update“ auf SQL-Ebene.

Abgesehen von den traditionellen datenbankseitigen Zugriffen durch SQL-Kommandos stehen Protokoll-Schnittstellen wie HTTP, WEBDAV (Web Distributed Authoring and Versioning) und FTP zur Verfügung, um Zugriffe von beliebigen Client-PCs ohne zusätzliche Installation zu ermöglichen. Dazu ist das Datenbankmodell eigens um Funktionen und Strukturen wie Verzeichnisse, Versionierung und ACLs (Access Control Lists) erweitert worden, die in einem Repository in der Datenbank liegen. Dieses Repository, das automatisch in der Datenbank zur Verfügung steht, bildet die hierarchische Verzeichnisstrukturen auf die XML-Ressourcen wie XML- Dateien oder Schemas intern ab. So wirkt ein Oracle- XML-Datenbank-Verzeichnis wie ein natives Betriebssystem-Verzeichnis. Der Anwender kann zum Beispiel Oracle-XML-

Datenbank-Verzeichnisse im Windows- Explorer öffnen und auf XML-Dateien zugreifen oder aber Web-Verzeichnisse im Internet-Explorer nutzen. Auch der FTP-Zugriff unterscheidet sich nicht von einem gewöhnlichen FTP-Server. Veränderungen, die über die Protokolle durchgeführt werden, können allerdings nur über den Austausch des gesamten Dokuments erfolgen.

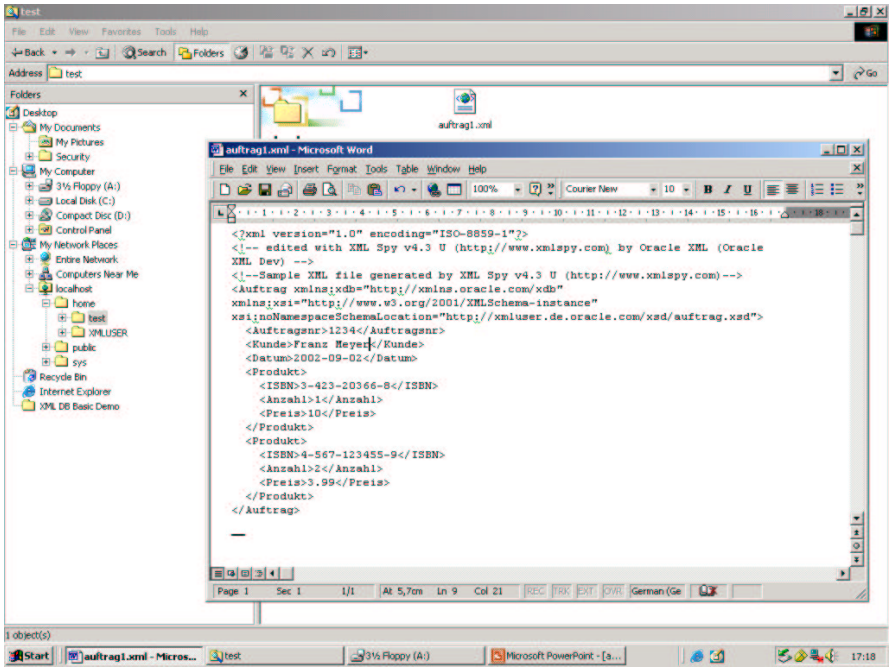


Abb. 3.1: Zugriff auf ein XML-Dokument über dem Windows-Explorer

Alle Dokumente, die auf diese Art und Weise geladen werden, sind automatisch geparst. Dies bedeutet, dass sobald die Datenbank anhand der URL das XML-Dokument als Instanz eines schon registrierten Schemas erkennt, wird das Dokument zerlegt und die Informationen in die entsprechenden Tabellen gespeichert. Sind zusätzliche Integritätsbedingungen wie Constraints und Trigger durch die üblichen Datenbankoperationen hinzugefügt worden, werden diese ebenfalls berücksichtigt.
Insbesondere beim Verstoß gegen ein Constraint kann dies für einen Ladevorgang über das FTP-Protokoll bedeuten, dass XML-Dokumente wegen Integritätsverletzung nicht geladen werden. Dies ermöglicht dem Anwendungsprogrammierer zusätzliche zentrale Funktionalität in die Datenbank zu verlegen.

```
ftp> open mucssu3 2100
Connected to mucssu3.
220 mucssu3 FTP Server (Oracle XML DB/Oracle9i Enterprise
 Edition Release 9.2.0.1.0 - Production) ready.
User (mucssu3:(none)): xmltest
331 pass required for XMLTEST
Password:
230 XMLTEST logged in
ftp> mput auftrag2.xml? y
200 PORT Command successful
150 ASCII Data Connection
550- Error Response
ORA-00604: error occurred at recursive SQL level 1
ORA-00001: unique constraint (XMLUSER.REFERENCE_IS_UNIQUE)
violated
550 End Error Response
ftp: 528 bytes sent in 0,00Seconds 528000,00Kbytes/sec
```

Abb. 3.2: Kopierversuch eines ungültigen XML-Dokuments

Ein weiterer wichtiger Aspekt beim Arbeiten mit XML-Dokumenten ist der Zugang durch das HTTP-Protokoll. Wie wir gesehen haben, unterstützt Oracle bei XML-Ressourcen, die in der Datenbank gespeichert sind, das HTTP- und WEBDAV-Protokoll.
Darüber hinaus ermöglicht ein Servlet, auf Tabellendaten über URLs zuzugreifen. Nur durch Angabe einer URL in einem Browser können somit ohne zusätzliche SQL-Programmierung, relationale Daten angezeigt werden. Das Servlet unterstützt das Generieren von XML- and Nicht-XML-Inhalte wie auch das Transformieren der Resultate durch Stylesheets. Dies erledigt ein Stylesheet-Prozessor, der direkt in der Datenbank liegt. Die Transformation kann durch SQL-Kommando oder durch den Aufruf einer URL, die den Inhalt des XML-Dokuments und des Stylesheets beinhaltet, durchgeführt werden. Im folgenden Beispiel (Abbildung 3.3.) wird durch das Wort „transform“ der Stylesheet-Prozessor aufgerufen. Als Resultat wird das transformierte Ergebnisdokument zurückgeliefert.

```
http://mucssu3:8080/oradb/XMLUSER/AUFTRAG_TAB/ROW[auftrag/kunde='Franz
Meyer'?transform=/public/xsl/auftrag.xsl&contenttype=text/html
```

Abb. 3.3: URL für ein zu transformierendes XML-Dokument

4 Fazit und Ausblick

Da die Oracle-Datenbank in der Lage ist, XML-Daten und relationale Daten in einem Server zu verwalten, ist es möglich, einerseits relationale Sichten auf XML-Dokumente und andererseits XML-Sichten auf relationale Daten zu ermöglichen.
Im ersten Fall kann unter Anwendung der XPATH-Technik und unter Zuhilfenahme von zusätzlichen Oracle-Funktionen, wie zum Beispiel der EXTRACT-Funktion eine relationale Sicht erzeugt werden. So bleibt für den Anwender völlig transparent, ob die

Daten relationalen oder XML-Ursprungs sind. Relationale Reporting-Werkzeuge können somit ohne Veränderungen auf die Daten zugreifen.
Der umgekehrte Weg, relationale Daten im XML-Format anzuzeigen, ist ebenfalls sehr wichtig, da eine große Menge von Business-Daten mittlerweile in Datenbanken relational gespeichert ist. Dabei wird die Nachfrage von Applikationen, diese Daten im XML-Format zur Verfügung zu stellen, immer größer. Oracle bietet für diesen Fall die Möglichkeit, SQL-Daten per Browser im XML-Format anzuzeigen. Auf diese Weise können nicht nur rein relationale Daten und sondern auch XML-Daten angezeigt werden.
Um komplexere XML-Strukturen zu erzeugen, unterstützt Oracle außerdem Abfragen im SQL-Stil. Diese Art der Abfragen, auch bekannt unter dem Namen SQL/XML, wird durch die SQLX-Gruppe (6), in der Oracle ein aktives Mitglied ist, standardisiert. SQL/XML ist eine Erweiterung zu SQL und führt neue Funktionen und Operatoren ein. Mit Funktionen wie zum Beispiel XMLElement, XMLAttributes in SQL-Abfragen, kann der Programmierer XML-Dokumente generieren, die beliebig mit anderen relationalen oder XML-Tabellen kombiniert werden können.
Die W3C Gruppe, arbeitet im Moment an der Standardisierung von XQuery (5), einer reinen XML-Abfragesprache. Da sich diese im Moment noch im Working Draft- Status befindet, bietet Oracle für interessierte Anwender einen XQuery–Prototypen im „Oracle Technology Network“ (4) zum Herunterladen an.

Mit der Oracle-XML-Datenbank Technologie ist ein großer Schritt zur Integration von relationalen Daten und XML-Dokumenten gemacht worden. Die Anschaffung von zusätzlichen XML-Servern sind somit nicht notwendig, um XML-Daten zu nutzen. Analysten sagen eine größere XML-Nutzung voraus; relationale Daten werden ihre Bedeutung aber beibehalten. Die Integration beider Welten in einem zentralen System, mit offenen standardkonformen Zugriffen, ist deshalb ein zukunftsweisender Technologieansatz.

4 Literaturverweise

(1)Bulletin Worldwide XML Database Management Software Forecast, 2002–2006 Analyst Susan Funke
(2)Introduction to Native XML Databases by Kimbro Staken October 31, 2001: http://www.xml.com/pub/a/2001/10/31/nativexmldb.hmtl
(3)Oracle9i XML Database Developer’s Guide – Oracle XML DB Release 2 (9.2) March 2002, Part No. A96620-01
(4)Oracle Technology Network: http://otn.oracle.com
(5)W3C Architecture Domain: http://www.w3.org/XML/Query
(6)SQLX Workgroup: http://www.sqlx.org
(7)Relational and Beyond: Carsten Czarski und Ulrike Schwinn: http://www.oracle.com/global/de/pub/knowhow/index.html?oracle_xml.html

Automatic Database Configuration for DB2 Universal Database: Compressing Years of Performance Expertise into Seconds of Execution

Eva Kwan
evakwan@ca.ibm.com

Sam Lightstone
light@ca.ibm.com

Berni Schiefer
schiefer@ca.ibm.com

Adam Storm
ajstorm@ca.ibm.com

Leanne Wu
leannewu@ca.ibm.com

IBM Canada, Toronto Laboratory

Abstract:

This paper describes the DB2® Configuration Advisor, an expert tool for the configuration of DB2 databases. This advisor has shown dramatic results for tuning and configuring DB2 servers on UNIX® and Windows® platforms. The recognition of the essential need for administration and design tools has spurred renewed interest among leading relational database management system (RDBMS) vendors. The DB2 Configuration Advisor is a key feature in DB2's Autonomic Computing self-managing technology portfolio. This paper discusses the purpose and features of this expert advisor. Experimental results are presented with a description of the advisor's interfaces.

1. Introduction and motivation

In this paper we present an overview and experimental results for a database configuration tool for DB2 Universal Database™ for UNIX and Windows (DB2), known as the DB2 Configuration Advisor. This advisor makes recommendations on initial settings for configuration parameters and memory allotment within the database management system, which can be adopted by inexperienced administrators or fine-tuned by more experienced personnel. The performance of a tuned configuration versus an untuned configuration may be dramatic, with a measurable and significant performance improvement. The DB2 Configuration Advisor is one of a growing set of features for DB2 that reduces the human expertise required in database tuning and administration by combining automation, artificial intelligence, and expert advice. The experimental results shown illustrate how this technology can greatly simplify expert tuning work.

Understanding the priorities of administrators is critical in reducing manual administration. The increased complexity and volume of data seen in modern database systems has increased the burden of database administrators. The database administrator now faces a diverse combination of tasks, as shown in Figure 1.

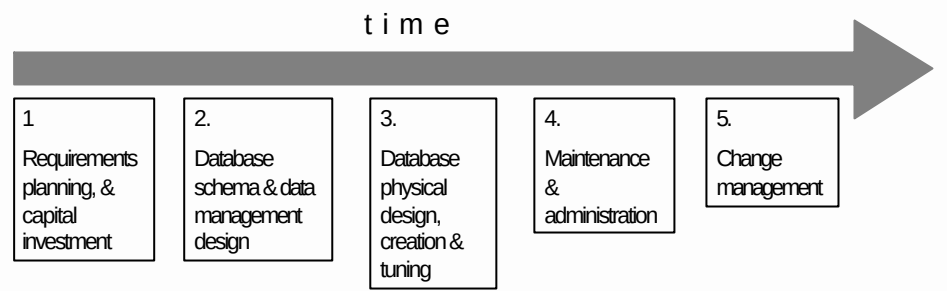


Figure 1. Tasks of a database administrator over the life cycle of a database

Often the design choices for databases are complex and nontrivial. Once a database has a physical and logical design, the operation of the database requires substantial human attention for numerous tasks including, but not limited to: table reorganization, data statistics collection, backup control, disaster recovery planning, performance tuning, and problem analysis.

As database management systems have grown in size and complexity, recognition of the importance of ease of administration and of better design tools has also increased, and this recognition has spurred renewed interest in research and development of software to reduce administration requirements [Be98] [Br94] [Ch99] [FST88]. IBM® has recently increased its focus on autonomic computing, which aims to produce self-managing systems that run with a minimum of human intervention. This research has resulted in IBM's Autonomic Computing initiative, which aims to enhance the IBM portfolio of clients, middleware, servers, and storage products through self-managing features, and which has the long term objective of producing true autonomic computing [IBM02a]. This initiative demonstrates IBM's commitment to providing solutions for increasingly complex systems. Within DB2, the Autonomic Computing initiative has as its objective the development of databases that will require minimal expertise and maintenance. As an automatic configuration feature, the Configuration Advisor falls under the aegis of IBM's corporate wide Autonomic Computing research and development effort, of which the DB2 Autonomic Computing effort is a part.

The goal is to create intelligent software tools that will reduce the burden on database administrators by providing expert design systems, performance tuning and configuration technology, with easier to use interfaces for administration and tools for automation. One area in which research has been focused has been in the development of internal tuning and configuring technology within the product deliverable. The DB2 UDB Configuration Advisor is one such product of this research. It allows large database servers to be configured for performance in seconds. While the utility has been available with DB2 for a few years, significant improvements have been performed for DB2 Universal Database version 8.1. IBM announced IBM DB2 Version 8 for Linux, UNIX and Windows on September 17, 2002, and the product was made generally available on Nov. 21, 2002.

2. Objectives and features

The Configuration Advisor is part of DB2 Universal Database for UNIX and Windows. IBM's DB2 Universal Database product provides a set of configuration parameters settings that can be used to configure and tune the DB2 environment. The objective of the Advisor is to define a set of initial database configuration parameters and memory assignments to optimize system performance without extensive monitoring of the system. These configuration parameters include ones that control memory distribution (sorting, locking, caching database pages and working heaps), parallelism, I/O optimization (asynchronous page readers and writers), many aspects of logging (file size, buffer size), and recovery. While there are default settings provided with each installation of DB2, it would be impossible to provide a single configuration that would perform equally well for the diverse workloads and systems that use DB2. The complexity of the memory topologies for enterprise database management systems makes these assignments a difficult task, even for skilled administrators deeply aware of their workload characteristics. The Configuration Advisor can be used both to recommend configuration settings, as well as to apply the settings.

The Configuration Advisor has been developed with three execution interfaces. The first is a graphical interface that is part of the DB2 Control Center, the graphical interface for DB2's administration. The second is through a DB2 system command, whereby the Configuration Advisor can be invoked through a text command [IBM02c]. The command level interface is a popular interface for DB2 administration, and also lends itself towards scripting. The third interface is through programmable APIs (DB2 provides both a C level API and a stored procedure interface to the Configuration Advisor) which allows database applications, including independent software vendors (ISVs), to programmatically invoke the Configuration advisor for seamless configuration of DB2 databases [IBM02b].

3. Design

The DB2 Configuration Advisor is designed on the principle that configuration choices can be made by modeling each database configuration setting as a mathematical expression which combines three sets of information regarding the system environment, the database characteristics and user priorities. The three sets of information are:

1. User specification of the database environment (designed as a small set of basic input, generally requiring minimal skill).
2. Autonomically sensed system characteristics (such as number of CPUs, disks, amount of RAM, number and size of relational tables, etc).
3. Expert heuristics for database configuration, as reported by experienced database administrators and performance tuning experts.

The first two of these information classes are essentially parametric, and can typically be represented as scalar values which are input by the user. The advisor is designed so that the responses collected from the user make up the minimal set of characteristics that are

required to effectively configure a database. Furthermore, the advisor is designed so that the questions require neither lengthy nor detailed analysis of the database in order to be answered. The second set of input information is obtained through the automatic detection of system resources, which includes the amount of physical memory, number of disks, storage available, and the number of CPUs. Included in this second set of information is information about the existing database's current characteristics. This information includes number of tables defined, the number and size of defined buffer pools, and the number of database containers. The first two classes of parametric information are then combined with a set of expert heuristics to define mathematical expressions of the estimated ideal settings, which define the configuration model. The expert heuristics were collected through an ad hoc process of surveying over a dozen of DB2's leading performance experts and architects, as well as through review of published information on DB2 performance tuning [SV99] [IBM02d].

The configuration model expresses the configuration settings as a mathematical expression. The model for the configuration settings is further divided into three distinct classes:

1. Independently modeled configuration settings, which can be modeled independent of other configuration settings.
2. Dependant configuration setting, where the value of one setting affects the model of another (or perhaps co-dependency), and
3. Zero sum game relationships (such as memory for sort, caching, locking etc) in which a fixed resource must be divided among a set of configuration parameters.

Figure 2 illustrates the design model for the Configuration Advisor. With one of the design goals of the advisor being to reduce human skill and involvement, the user specification is kept intentionally trivial, with a long term aim of reducing it further.

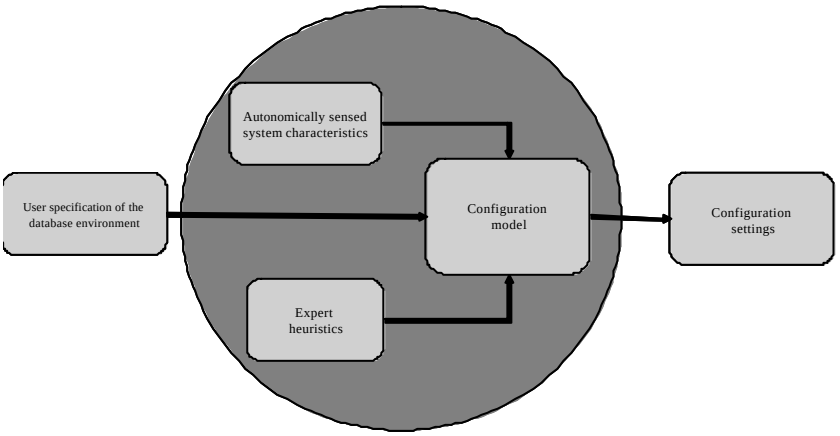


Figure 2: Design model for the DB2 Configuration Advisor combining user specified basic description, autonomically sensed system characteristics and expert heuristics.

The configuration model includes algorithms that are based on documented guidelines, expert advice, and system behaviors generalized from analysis of workloads. For most configuration parameters, the advisor automates the calculation of settings based on known guidelines. Other configuration parameters are more complex, and this is where the advisor incorporates the expert heuristics of performance experts and architects at IBM. The developers have an in-depth insight into DB2's implementation, and performance experts have valuable experience in performance tuning.

The goal of the advisor is to be useful across a wide range of systems so its internal algorithms also include information gleaned from analysis of a wide range of workloads. The analysis allows the advisor to better classify systems and choose the key characteristics on which to base recommendations, a difficult task due to the complex interactions and uniqueness of each database. The result, as our results indicate, is a good set of initial settings, which can later be fine-tuned as administrators monitor system performance during run time.

To the authors' knowledge this approach is unique in the industry. Several other vendors provide database configuration tools; however these tools typically focus on a distinct subset of configuration settings (such as buffer pool configuration), are not generally based on parametric models of database configuration, and typically lack the extensive autonomic sensing capability that the DB2 Configuration Advisor embodies.

4. Experimental results

To examine the effectiveness of the Configuration Advisor, we performed four experiments on systems running distinct workloads and environments, comparing the performance of the database after tuning by the Configuration Advisor to the system performance achieved through tuning by an expert database administrator. These experiments include a degree of inaccuracy, since the tuning performed by human administrators has variable quality, and it is impossible to assess how close this tuning is to optimal. However, what these experiments do illustrate is the degree to which the Configuration Advisor is able to tune the database system relative to the human expert. While the human administrator typically spends a number of days tuning these configuration parameters, the advisor provides recommendations for the same set of parameters within seconds.

The four experiments included:

- An Online Transaction Processing (OLTP) industry standard workload, run once on a 32-bit implementation, and again on a 64-bit implementation. This benchmark simulates a population of terminal operators executing transactions against a relational database. The benchmark models the transaction environment of an order-entry environment.
- Two operational workloads tested on-site at two of the world's leading global investment banks.

While all tests in this set of experiments were run on AIX, this was more due to general availability of test systems rather than any platform-specific constraints on the advisor.

4.1 64-bit OLTP

The 64-bit OLTP experiment was run on an RS/6000® 44P Model 270 4-way 375 MHz Power3-II server, configured for 2-way processing. The system had 8 GB memory, and was running AIX 4.3.3 with DB2 UDB EE V7.2 (64-bit). The storage system on this server included 3 ServeRAID 4H (SCSI) adapters as follows: fifty-six 9 GB SCSI Disks (data), fifty-six 9 GB SCSI Disks (data), fourteen 9 GB SCSI Disks (log, backup, temp tablespaces). The hand-tuned system was configured by DB2 performance specialists at IBM. In this experiment the database performance after tuning with the Configuration Advisor was observed to be 93.58% of the hand-tuned system.

4.2 32-bit OLTP

The 32-bit OLTP experiment was run on an RS/6000 44P Model 270 4-way 375 MHz Power3-II server, with 8 GB of memory, running AIX 4.3.3 with DB2 UDB EE V7.2 (32-bit). The storage system on this server included 3 ServeRAID 4H (SCSI) adapters as follows: twenty-eight 9 GB SCSI Disks (data), forty-two 9 GB SCSI Disks (data), fourteen 9 GB SCSI Disks (log, backup, temp tablespaces). As in the 64-bit experiment, the hand-tuned system was configured by DB2 performance specialists. In this experiment the database performance after tuning with the Configuration Advisor was observed to be 91.52% of the hand-tuned system.

4.3 Global investment bank A

Two sets of tests were conducted on production workloads. The first of these tests was conducted on a high volume OLTP test system that stress tested the database using an authentication application. The application simulated authentication operations that would result from customer accesses of the bank's products.

The test environment consisted of a 6-way RS/6000 Model F80 server with each processor running at 500 MHz. The server had 4 GB of memory and was running AIX 4.3.3. Its data was spread over 14 of its 16 drives (two 9 GB drives and twelve 18 GB drives). In this experiment the hand-tuned system was configured by the bank's database administrator. The results of the experiment show that after the Configuration Advisor was run, the workload performed 5.62% better than the hand-tuned system.

4.4 Global investment bank B

The second of the two production system tests was conducted on a system that allows customers to view their web pages and change their preferences. These preferences are sequenced into records in a DB2 UDB database. The transaction types are mixed between selects and inserts or deletes, with 60% of the workload being selects. To

provide for load balancing and for hot backup solutions, the system exploits two servers with peer-to-peer replication.

The two servers had identical hardware specifications and topologies. Both were IBM RS/6000 Model 270 4-way 375 MHz servers with 4 GB of memory running AIX 4.3.3. The storage system included sixteen 18 GB drives. At the time of the experiment, the research prototype of the Configuration Advisor did not include a model for systems that deploy peer-to-peer bulk replication. As a result, the advisor under configured memory for locking and logging, resulting in sub optimal system performance. The administrator at the customer site decided to leave the settings for locking and logging at their pre-advisor values and adopted all of the remaining Configuration Advisor recommendations. The result was a best-ever system performance for the database, with the new configuration performing at 224.72% of the original administrator-tuned throughput. It is interesting to note that the alteration in buffer pool size (set by the advisor) was an important change in the configuration. While the advisor was clearly inadequate without the planned extensions to support bulk replication, the recommendations of the advisor were effective in combination with two adjustments by the database administrator (for locking and logging) in achieving a massive improvement in system performance.

4.5 Experimental Summary

In all of these experiments (the result of which are in Figure 3) the Configuration Advisor was able to evaluate and compute a revised or proposed configuration in two or three seconds. The performance of the advisor is significant when compared to the amount of time typically spent configuring large database management systems, usually on the order of one to two weeks.

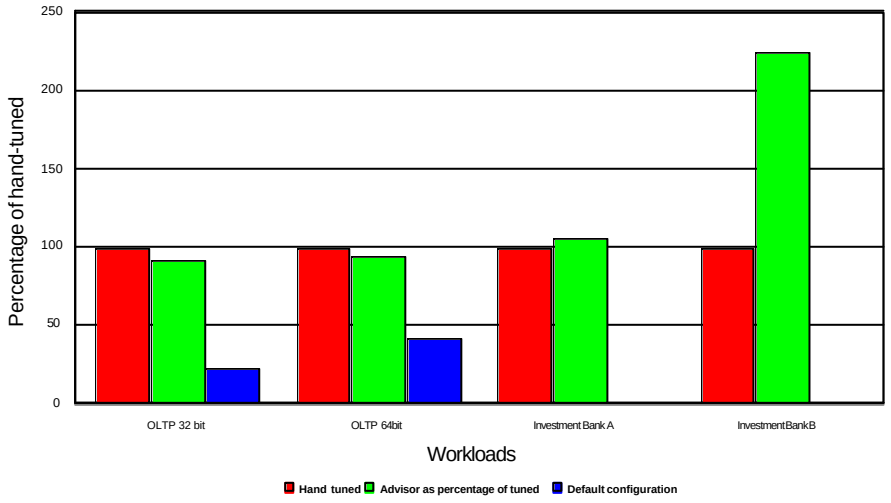


Figure 3. Database performance for hand-tuned parameters versus Configuration Advisor tuning

In the two cases in which the advisor was tested against systems configured by IBM performance experts, the resulting performance was within 10% of the hand-tuned result. As well, in these two cases when the Configuration Advisor selections were tested against the default configuration, the Advisor significantly outperformed the default settings.

In the two cases where the Configuration Advisor was tested against databases in the field, the Configuration Advisor exceeded the hand-tuned result (though in the case of "Global investment bank B" two parameters were left at the hand-tuned setting to account for the special needs of peer-to-peer replication).

5. Conclusions

The Configuration Advisor has the ability to reduce and possibly eliminate the tedious and time-consuming task of configuring a system for desired performance as is shown with the benchmarking results. In this paper we have studied the quality of the recommendations provided by the Configuration Advisor by comparing its performance to both industry standard benchmark systems tuned and configured by database specialists inside IBM, and systems in use by two major brokerage firms. Our results indicate that the advisor shows promise for providing quality recommendations on database configuration, and, in some cases, exceeding the performance quality of human-configured systems, as observed in the case of the two investment banking systems studied. If performance on the system is not an absolute priority, the Configuration Advisor eliminates the need for frequent manual tuning of performance related configuration parameters. If the system is required to be configured for near optimal performance, the advisor supplies a configuration that can serve as a springboard for further fine-tuning based on specific workload characteristics.

The simplicity and speed of the Configuration Advisor provide compelling arguments for deployment, and as part of both the Autonomic Computing initiative the advisor demonstrates some of the benefits that autonomic computing promises to bring. All these factors combine to make the Configuration Advisor a valuable tool for database administrators.

6. Future work

In addition to further refinement of the current modeling for database configuration, there are several avenues for future research. The first is incorporating automatic workload characterization schemes into the inputs that are used by the Configuration Advisor. Currently, the Advisor depends on user-specified workload characterizations. If the advisor could automatically detect these characteristics the result would be less work on the part of the user and increased Advisor accuracy. Another area in which further work could be performed is in investigating closed-loop performance feedback for the advisor. The advisor's current architecture produces identical results regardless of the number of times it is run, if the same inputs are used on a given system. By providing a mechanism by which the advisor could learn about the effectiveness of its past

recommendations, it will be possible for the advisor to provide a better and more intelligent configuration with subsequent iterations. Additionally, there is ongoing research to refine the configuration model within the advisor. Improving this model will allow the advisor to perform well on an even wider range of database systems.

References

[Be98] Bernstein, M. et. al.: The Asilomar Report on Database Research. ACM SIGMOD Record 27(4), September, 1998; pp. 74-80.

[Br94] Brown, K. et. al.: Towards Automated Performance Tuning For Complex Workloads. Proceedings of 20th International Conference on Very Large Databases, Santiago de Chile, Chile, 1994. Morgan Kaufmann, 1994; pp 72-84.

[Ch99] Chaudhuri, S. et. al.: Self-Tuning Technology in Microsoft SQL Server. IEEE Data Engineering Bulletin 22(2), June 1999; pp. 20-26.

[FST88] Finkelstein, S.; Schkolnick, M.; Tiberio, P.: Physical Database Design for Relational Databases. ACM Transactions on Database Systems 13(1), March 1988; pp. 91-128.

[IBM02a] IBM Research: Autonomic Computing. Online: `http://www.research.ibm.com/autonomic/`

[IBM02b] IBM Software: DB2 Information Center: db2AutoConfig – Autoconfigure. Online: `http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v8infocenter.d2w/report?target=mainFrame&fn=r0003294.htm`

[IBM02c] IBM Software: IBM DB2 Universal Database Version 8 Command Reference. Online: `ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/pdf/letter/db2n0e80.pdf`

[IBM02d] IBM Software: IBM DB2 Universal Database Version 8 Administration Guide: Performance. Online: `ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/pdf/letter/db2d3e80.pdf`

[SV99] Schiefer, B.; Valentin, G.: DB2 Universal Database Performance Tuning. IEEE Data Engineering Bulletin 22(2), June 1999; pp. 12-19.

Notices

The following are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX	DB2
IBM	Universal Database

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Logische Datenmodellierung zur Abbildung mehrdimensionaler Datenstrukturen im SAP Business Information Warehouse

Michael Hahne

cundus AG
Freiherr-vom-Stein-Straße 13a
D-55559 Bretzenheim
Michael.Hahne@cundus.de

Abstract: Innovative technische Konzepte des Data Warehousing, OLAP und Data Mining begleiten eine zunehmende strategische Ausrichtung der Informationsverarbeitung mit einer stärkeren Fokussierung auf Aspekte der Analyse und Entscheidungsunterstützung. In diesem Segment positioniert die SAP AG ihre Lösung mySAP Business Intelligence, die auf dem aktuellen Release des Business Information Warehouse (BW) als technologischer Kernkomponente basiert. Das BW ermöglicht den Aufbau von Systemen mit einer konsistenten und einheitlichen Datenbasis für die vielfältigen betrieblichen Anwendungen von Fach- und Führungskräften. Die Leistungsfähigkeit und erfolgreiche Nutzung solcher Systeme werden maßgeblich durch die Möglichkeiten der Modellierung und die Mächtigkeit des zu Grunde liegenden Datenmodells bestimmt. Hierarchischen Dimensionsstrukturen kommt dabei eine zentrale Rolle zu, da die hierdurch festgelegten Konsolidierungspfade die Grundlage für die Navigation in analytischen Datenbeständen sowie zahlreiche OLAP-Operationen bilden. Die Abbildung dieser Strukturbestandteile mehrdimensionaler Datenmodelle im BW ist durch Variantenreichtum und innovative Ansätze gekennzeichnet und wird in diesem Artikel eingehend betrachtet.

1 Einleitung

Mit dem *Business Information Warehouse (BW)* ist die SAP AG in den Markt für Business Intelligence eingestiegen. Besonderes Leistungsmerkmal des BW ist der *Business Content* als eine Sammlung vorgefertigter betriebswirtschaftlicher Lösungen, die im BW neben den umfassenden Funktionalitäten zum Aufbau und Betrieb eines Data Warehouse zur Verfügung gestellt werden. Besonderes Augenmerk haben dabei Systemumgebungen, in denen SAP R/3 als vorgelagertes Quellsystem vorhanden ist.
Unter dem Begriff *mySAP Business Intelligence* sind Anwendungskomponenten und Dienstleistungen für die Entscheidungsunterstützung in Unternehmen zusammengefasst, deren Hauptkomponenten das Business Information Warehouse als Basistechnologie zur

Datenhaltung ist. Weitere Bestandteile sind u. a. der Business Explorer zur Datenauswertung und das Enterprise Portal als zentraler Zugangspunkt mit Single-Sign-On-Funktionalität zur Verteilung von Berichten und Informationen. Das Strategic Enterprise Management, neuerdings Bestandteil der Lösung mySAP Financials beinhaltet Funktionen für Unternehmensplanung, Performance Management und Konsolidierung. Die Datenhaltungskomponente des SEM ist ebenfalls das BW.[1]

Für viele Lösungen aus dem Hause SAP ist das Business Information Warehouse die wesentliche technologische Basiskomponente zur Datenspeicherung, auf deren Basis unterschiedlichste analytische Anwendungen wie beispielsweise die Bewertung und Optimierung von Logistikketten oder Kundenbeziehungen aufbauen. Einen signifikanten Einfluss auf die Leistungsfähigkeit und Akzeptanz solcher Systeme hat die Modellierung, deren Möglichkeiten durch die Grenzen des zu Grunde liegenden Modells limitiert sind.

Als zentrales Charakteristikum gewährleisten multidimensionale Sichtweisen auf unternehmensinterne und -externe Datenbestände brauchbare Näherungen an das mentale Unternehmensbild des Managers, denn in der Tat untersuchen Manager betriebswirtschaftlich relevante Sachverhalte unter Berücksichtigung zahlreicher Einflussfaktoren. So wird beispielsweise nach dem Absatz einer Produktgruppe in bestimmten Verkaufsregionen über einen definierten Zeitraum hinweg gefragt. Die Anordnung betriebswirtschaftlicher Variablen bzw. Kennzahlen, wie z. B. Umsatz oder Kostengrößen, erfolgt entlang unterschiedlicher Dimensionen, wie z. B. Kunden, Artikel und Regionen, und diese Strukturierung gilt als geeignete entscheidungsorientierte Sichtweise auf betriebswirtschaftliche Tatbestände. Bildlich gesprochen werden die quantitativen Kenngrößen in mehrdimensionalen Würfeln gespeichert, deren Kanten durch die einzelnen Dimensionen definiert und beschriftet sind.

In Abschnitt 2 erfolgt die Darstellung der Architektur und des spezifischen Datenmodells des Business Information Warehouse. Auf die Möglichkeiten der Abbildung hierarchischer Strukturen mit den Mitteln dieses Modells wird anschließend in Abschnitt 3 eingegangen. Die Zusammenfassung der gewonnenen Arbeitsergebnisse erfolgt in Abschnitt 4.

2 Business Information Warehouse

Die Architektur des Business Information Warehouse ist angelehnt an die allgemeinen Referenzarchitektur für ein Data Warehouse. Ausgehend von verschiedenen möglichen Quellsystemen erfolgt ein im allgemeinen periodischer Datenladevorgang in die zentrale Datenhaltung des Business Information Warehouse (BW). R/3-Systeme haben eine besondere Bedeutung als vorgelagertes OLTP-System, aber auch die Anbindung von R/2-, Non-SAP- und anderen BW-Systemen ist neben dem Import über flache Dateien möglich.

Die Verwaltung des BW-Systems erfolgt in der Administrator Workbench, einer Administrationskonsole, in der die Steuerung, Überwachung und Pflege des gesamten Extraktions- und Lademanagements von Quelldaten, die Benutzerverwaltung sowie die Modellierung der Datenstrukturen im BW erfolgt. Im Mittelpunkt der Architektur steht der

[1] Die Gruppierung und Begriffsbildung ändert sich bei der SAP AG häufiger.

eigentliche Business Information Warehouse-Server, der für die Ablage, Aufbereitung und Abfrage der Daten im BW verantwortlich ist. Die wesentlichen Objekte zur Datenablage sind dem Datenfluss folgend zunächst der Persistant Staging Area (PSA), der Operational Data Store (ODS) und die Info-Cubes. Während der PSA temporären Charakter hat, sind die Daten im ODS und in den Info-Cubes für die dauerhafte Ablage gedacht. In den relational strukturierten ODS-Objekten erfolgt die Speicherung von belegnaher Information, die Info-Cubes bilden die mehrdimensionalen Strukturen als Grundlage für OLAP-Analysen ab.
Basis für Auswertungen und die Erstellung von Berichten mit verschiedenen Front End-Werkzeugen sind die im Business Information Warehouse abgelegten Datenstrukturen. Mögliche Kategorien von Anwendungen basieren zum einen auf Werkzeugen zur Unterstützung des Managements, die von einfachen Berichts- und Abfragesystemen über leistungsstarke Werkzeuge zur Entscheidungsunterstützung bis zu den Führungsinformationssystemen reichen. Zum anderen handelt es sich um die unter den Stichworten Data Mining und OLAP diskutierten Anwendungsbereiche. Eine einheitliche Metadatenbasis ist im allgemeinen für alle auf das Business Information Warehouse zugreifenden Front-End-Werkzeugen essentiell.
Eine grobe Übersicht der Architektur und des Zusammenspiels der beteiligten Komponenten des Business Information Warehouse ergibt sich aus Abb. 1.

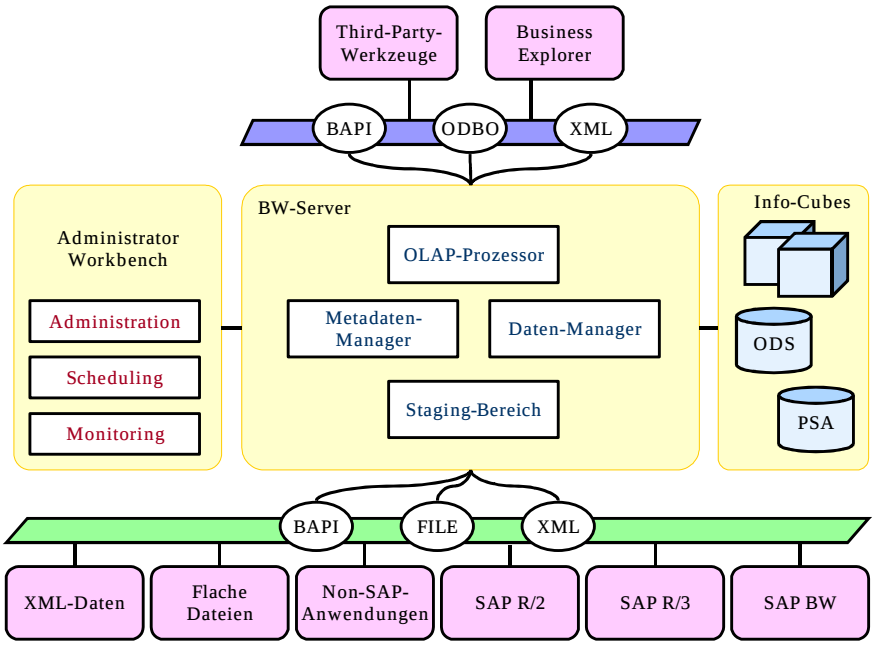


Abb. 1: Architektur des Business Information Warehouse 3.0

Integraler Bestandteil des Business Information Warehouse sind die Werkzeuge Business Explorer Analyzer als Excel-basiertes OLAP-Werkzeug und Crystal Reports für formatierte druckoptimierte Berichte.[2] Der Bereich des Webreportings wird durch die neue Technologie auf Basis des Web Application Servers und dem Web Application Designer adressiert. Das zentrale Front-End für SAP BW ist der Business Explorer (BEx). Dieser besteht seit der Version 3.0 des BW aus den Komponenten BEx Analyzer, BEx Browser, BEx Query Designer und dem BEx Web Application Designer.
Basis zur Speicherung der Daten im Business Information Warehouse bildet die relationale Datenbanktechnik. Für die Speicherung mehrdimensionaler Datenstrukturen in relationalen Datenbanken hat sich das Star Schema als Standard durchgesetzt [Ha99]. Zunächst erfolgt daher in Abschnitt 2.1 eine kurze Darstellung mehrdimensionaler Datenstrukturen und deren Abbildung in Star Schema-Modellen. Das spezifische Datenmodell, auf dem das Business Information Warehouse basiert, wird anschließend in Abschnitt 2.2 behandelt. Die Diskussion der besonderen Eigenschaften und Leistungspotentiale dieses Modells erfolgt in den Abschnitten 2.3 und 2.4.

2.1 Mehrdimensionale Datenstrukturen im Star Schema

Fundamentales Paradigma eines Data Warehouse ist die Mehrdimensionalität, die abgelegten Informationen basieren auf mehrdimensionalen Datenstrukturen. Diesem Ansatz folgend werden Datenwerte in einem mehrdimensionalen Datenwürfel aufgespannt betrachtet, deren Zellinhalte die abgelegten Daten darstellen. Die Würfelkanten, im Sprachgebrauch mehrdimensionaler Datenstrukturen Dimensionen genannt, gliedern diese Werte auf und bilden eine wesentliche Grundlage für Analysemöglichkeiten. Betriebswirtschaftliche Kennzahlen wie beispielsweise Umsatz werden entlang dieser Dimensionen als Aufgliederungsrichtung, so z. B. über Kunden, Artikel und Regionen, betrachtet.
Die Dimensionselemente entlang einer Kante des Würfels lassen sich in einzelne Gruppen zusammenfassen. In einer Produktdimension ist beispielsweise eine Gruppierung zu Basisprodukten und Produktgruppen denkbar. Solch eine Gruppierung basiert auf einer 1:n-Beziehung zwischen Produktgruppen und Basiselementen. Mehrere derartige Beziehungen bilden die Konsolidierungs- oder Navigationspfade in der Dimension und formen die Dimensionsstruktur. Die Basiselemente auf der untersten Ebene der Struktur definieren die Granularität des Würfels und damit die detaillierteste zur Verfügung stehende Informationsebene. Die anderen Dimensionselemente werden als abgeleitete oder verdichtete Elemente bezeichnet. Zusammenfassend sind die verschiedenen Bestandteile, die eine Dimensionsstruktur formen, in Abb. 2 veranschaulicht.
Typische Dimensionsstrukturen sind balancierte Baumstrukturen wie in Abb. 2 dargestellt, die dadurch gekennzeichnet sind, dass die Basiselemente alle über gleich viele Stufen bis zu einem obersten Konsolidierungselement verbunden sind und dass es genau ein oberstes (Wurzel-)Element gibt.

[2] Crystal Reports ist komplett in die Werkzeuge des Business Information Warehouse integriert, insbesondere sind die speziellen Bedürfnisse des Listen-Reportings im Query Designer des BW integriert. Lizenzrechtlich ist das Produkt aber gesondert zu betrachten, ein unternehmensweiter Einsatz ist durch die my SAP-Lizenz nicht mit abgedeckt.

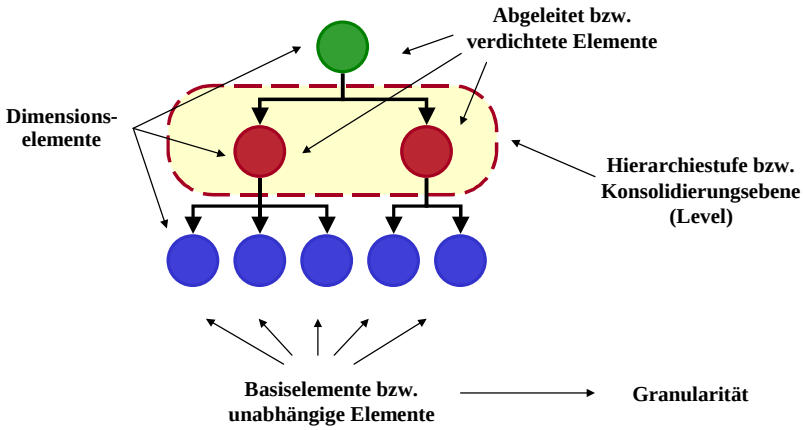


Abb. 2: Strukturbestandteile in Dimensionen

Die balancierte Dimensionshierarchie ist eine idealtypische Strukturformen, neben der es weitere Formen gibt. Eine ebenfalls häufig anzutreffende Struktur bilden parallele Hierarchien, in denen Basiselemente auf unterschiedlichen Wegen konsolidierbar sind, beispielsweise eine Kundendimension, in der die Gruppierung von Kunden zum einen regional und zum anderen nach sachlogischen Aspekten erfolgen kann. Statt von parallelen Hierarchien wird oftmals auch von verschiedenen Hierarchien in einer Dimension gesprochen.

Strukturen, bei denen die Bedingung einer 1:n-Beziehung der Ebenen in einer Dimension fallengelassen wird, stellen eine andere Komplexität dar, die besonderer Berücksichtigung bedarf. Die Zuordnung von Kalenderwochen zu Monaten ist ein Beispiel hierfür, da nicht alle Kalenderwochen eindeutig genau einem Monat zugeordnet sind [Ha02]. In diesen Fällen ist ein besonderes Augenmerk auf die Verdichtungsregeln zu legen. In einer Dimensionsstruktur, die z. B. Zuordnungen von Tochtergesellschaften zu Muttergesellschaften innerhalb einer Holding abbildet, sind die Anteilsverhältnisse bei der Konsolidierung zu berücksichtigen. Diese Formen von Hierarchien mit m:n-Beziehungen zwischen einzelnen Dimensionsebenen stehen unter dem Begriff der anteiligen Hierarchie in der Diskussion.

Eine weitere Bedingung, die beim Aufbau von Dimensionsstrukturen falen gelassen werden kann, ist die Eigenschaft der Ausgeglichenheit, die beschreibt, dass alle Dimensionselemente auf unterster Ebene über gleich lange Wege bis zu einem obersten Wurzelelement führen. Ein typisches Beispiel für eine unbalancierte Struktur ist eine Kostenstellenhierarchie, bei der eine eindeutige Zuordnung der Dimensionselemente zu den Konsolidierungsgruppen anhand der Navigationsschritte von den Basiselementen bis zum Wurzelelement nicht möglich ist (siehe Abb. 3).

Zur Abbildung mehrdimensionaler Datenstrukturen in relationalen Datenbanken sind verschiedene Ansätze eingeführt worden, die im wesentlichen unter dem Begriff *Star Schema* zusammengefasst und diskutiert werden [Ha99].

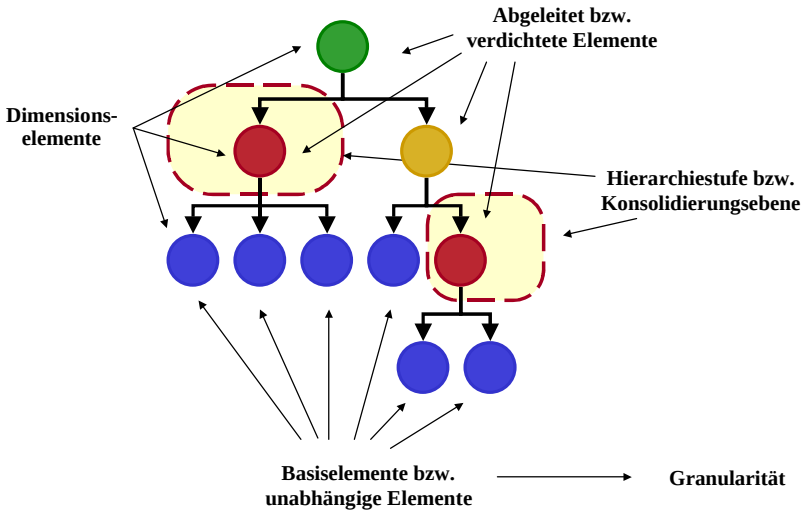


Abb. 3: unbalancierte Dimensionsstruktur

Die Faktentabelle mit den Kennzahlenwerten bildet den Mittelpunkt eines Star Schemas, um den herum die Dimensionstabellen als Satellitentabellen angeordnet sind [Nu98a]. Der identifizierende Schlüssel in der Faktentabelle ist aus unterschiedlichen Komponenten zusammengesetzt, die auf die Dimensionstabellen verweisen. Während in der Faktentabelle die Bewegungsdaten enthalten sind, beinhalten die Dimensionstabellen die Stammdaten, welche die Bewegungsdaten beschreiben. Sie definieren Suchkriterien und legen entlang der Hierarchien die Verdichtungsstufen fest [Nu98b]. Dimensionstabellen im Star Schema sind bei dieser Struktur im allgemeinen denormalisiert [Gl97]. Weitergehende Varianten des Star Schemas lassen diese einschränkende Bedingung fallen und es entstehen Modelle, die als Snow Flake-Schema bezeichnet werden.
Insbesondere aus Anforderungen an ein annähernd gleich bleibendes Antwortzeitverhalten heraus entstanden Modellvarianten, die diesen Aspekt besser berücksichtigen. Zentraler Aspekt ist dabei die Vorberechnung von verdichteten Werten, die beispielsweise in eigenen Aggregattabellen vorgehalten werden [Ha99].
Der Ansatz des Star Schemas zur Abbildung mehrdimensionaler Datenstrukturen in relationalen Datenbanken ist im wesentlichen auch die Grundlage des Datenmodells, das im Business Information Warehouse implementiert ist. Dieses wird im nächsten Abschnitt dargestellt.

2.2 Das erweiterte Star Schema der SAP

Die Datenhaltung des Business Information Warehouse basiert ebenfalls auf der relationalen Datenbanktechnik, die Speicherung mehrdimensional strukturierter Informationen folgt dem Ansatz eines Star Schemas, wobei zur Verbesserung in einigen Schwachpunk-

ten dieser Modellansatz entsprechend modifiziert wurde.[3] Wesentliche Verbesserungspotentiale ergeben sich dabei in der Behandlung von m:n-Beziehungen, von unblancierten Hierarchien sowie dem Umgang mit strukturellen Änderungen in Dimensionen und der Behandlung von Dateneingaben auf unterschiedlichen Verdichtungsstufen, wie sie beispielsweise bei der Ablage von Ist-Daten auf Tagesebene und von Plan-Daten auf Jahresebene auftreten.
Das zentrale Objekt zur Speicherung mehrdimensionaler Daten im BW ist der Info-Cube. Das Grundkonzept sieht dabei vor, dass die Informationen aus den Dimensionen für mehrere Info-Cubes gemeinsam genutzt werden, die Stammdaten somit übergreifend gültig sind [Sa00a].
Auch im erweiterten Star Schema bildet die Faktentabelle den Mittelpunkt des Modells. Die Fakten der Faktentabelle heißen im BW auch Kennzahlen oder Key Figures (z.B. Umsatz oder Absatz). Die Faktentabelle ist von Dimensionen umgeben, die aus der Dimensionstabelle und den Stammdatentabellen bestehen, exemplarisch ist dies in Abb. 4 für ein Modell mit den drei Dimensionen Kunde, Produkt und Zeit dargestellt.

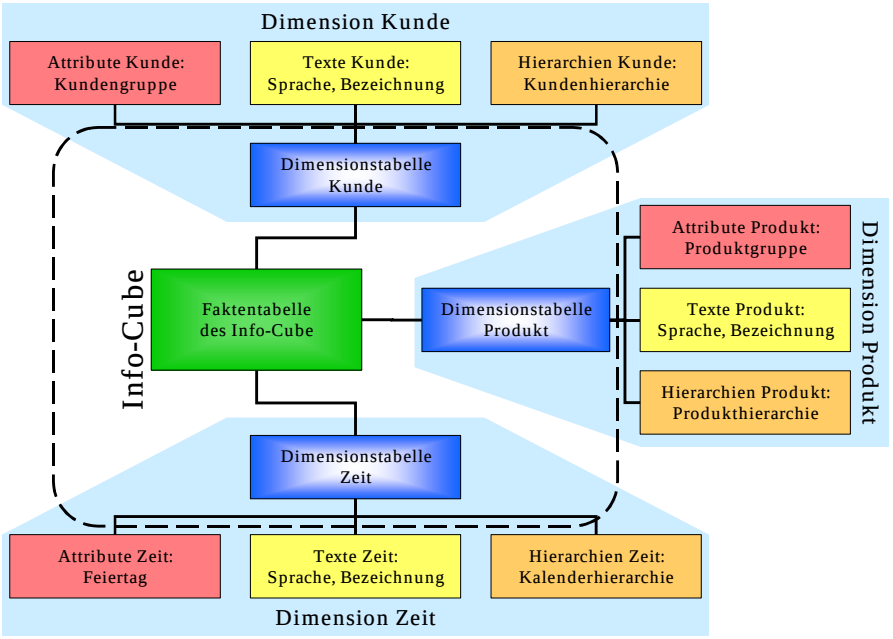


Abb. 4: Erweitertes Star Schema im SAP BW

[3] Das erweiterte Star Schema im BW ist gekennzeichnet durch eine große Anzahl von Tabellen, über die die Informationen verteilt abgelegt sind, und ist gegenüber klassischen Ansätzen aufgrund dieser Verteilung deutlich komplexer.

Der Begriff der Dimensionstabelle ist im BW mit einer besonderen Bedeutung versehen. Im erweiterten Star Schema bildet die Dimensionstabelle den Verknüpfungspunkt von der Faktentabelle zu den abgekoppelten Stammdatentabellen, in denen die eigentlichen Dimensionswerte und deren beschreibenden Attribute abgelegt sind.[4]
Die Basisinformationsbausteine im Business Information Warehouse sind die Info-Objekte. Sie werden global im BW-System definiert und sind über ihren technischen Namen eindeutig identifizierbar. Die im BW-Modell differenzierten betriebswirtschaftlichen Auswertungsobjekte sind zum einen die Merkmale, die den allgemeinen Dimensionselementen entsprechen, und die Kennzahlen, mit denen die Fakten beschrieben werden. Merkmale (Characteristics) und Kennzahlen (Key Figures) basieren beide auf dem Grundbaustein eines Info-Objektes, durch das damit Stamm- und Bewegungsdaten in strukturierter Form abgebildet werden. Auch die weiteren Objekte des BW basieren auf den Info-Objekten als elementare Komponenten, so sind auch die Felder in ODS-Objekten oder die Definitionen von Info-Sources durch sie beschrieben.

2.3 Stammdatenkonzept im BW

Merkmale als Dimensionselemente sind oftmals als numerischer Schlüssel definiert, beispielsweise die Artikelnummer aus einem vorgelagerten ERP-System wie z. B. R/3, zu deren weiterer Beschreibung ein Bezeichner gehört, im Beispiel der Artikelname. Die Ablage dieser beschreibenden Texte von Merkmalen erfolgt im BW in einer separaten Texttabelle, wobei die Texte in verschiedenen im System definierten Sprachen verfasst sein können. Beispielsweise sind zu einem Info-Objekt, das einen Länderschlüssel darstellt, mehrsprachige Landesbezeichnungen abbildbar.
Die Berücksichtigung von Dimensionsattributen ist im erweiterten Star Schema über Stammdaten-Attribute eines Merkmals gewährleistet. Diese werden in einer separaten Stammdatentabelle eines Merkmales gespeichert. Im BW-Sprachgebrauch wird allgemein von Attributen (z. B. Materialgruppe) gesprochen, wobei zwischen reinen Anzeigeattributen und Navigationsattributen zu differenzieren ist. Bei der Definition eines Info-Objektes wird festgelegt, ob überhaupt Attribute definierbar sind. Ist dies der Fall, wird von einem stammdatentragenden Merkmal gesprochen. Sowohl Anzeige- als auch Navigationsattribute beziehen sich auf ein stammdatentragenden Merkmal, aber nur Navigationsattribute sind losgelöst von diesen in Berichten darstellbar. Anzeigeattribute als ergänzende Information zu einem Merkmal können nur zusammen mit diesem in Berichten verwendet werden.
Hierarchien eines Merkmals, z. B. eine Länderhierarchie auf Basis des Info-Objektes Länderschlüssel, können in separaten Hierarchietabellen gespeichert werden. Diese Hierarchien werden Externe Hierarchien genannt (z. B. Standard-Kostenstellenhierarchie aus dem R/3-CO für das Merkmal Kostenstelle) und stellen vordefinierte Konsolidierungspfade dar. Die Benutzung des Begriffes Hierarchie ist im BW potenziell missver-

[4] Die Ablage der Informationen, die in Dimensionstabellen eines allgemeinen Star Schemas abgelegt sind, erfolgt im BW in den sog. Stammdaten-Tabellen, die eigentliche Dimensionstabelle im BW bildet nur die Schnittstelle von der Faktentabelle zu den verschiedenen Tabellen, in denen die eigentlichen Dimensionsinformationen abgelegt sind. Demzufolge korrespondiert der allgemeine Begriff der Dimensionstabelle im erweiterten Star Schema mit der eigentlichen Dimensionstabelle sowie den Tabellen für die Stammdaten-Attribute, die mehrsprachigen Texte und die Externen Hierarchien.

ständlich, denn im normalen Verständnis ist eine Hierarchie eine Sequenz von rekursiven Beziehungen zwischen Merkmalen. Demzufolge gibt es Hierarchien sowohl in der Dimensionstabelle über Merkmale, in der Stammdatentabelle über Attribute als auch in der eigentlichen Hierarchietabelle [Sa00a].
Die Stammdatentabellen sind nicht direkt an die Info-Cubes gebunden, die Verbindung erfolgt über SID-Tabellen, die mit künstlichen Schlüsseln arbeiten (Surrogate Ids). Über die SID-Tabellen, die Zeiger- oder Übersetzungstabellen, werden die Stammdatentabellen mit den Info-Cubes verbunden (siehe Abb. 5). Hierdurch können auch Merkmale auf Basis einer m:n-Beziehung wie z. B. Material und Farbe in einer Dimension abgebildet werden.

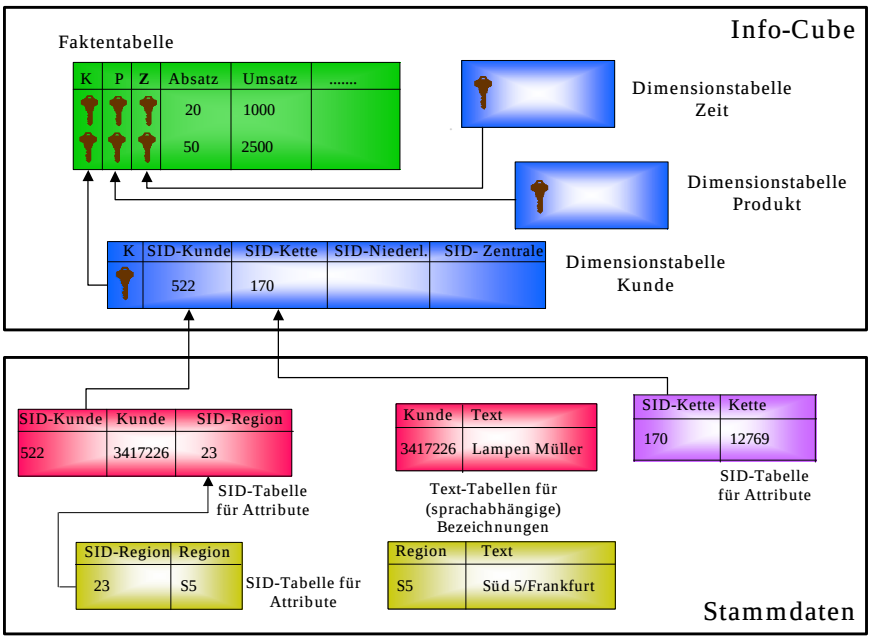


Abb. 5: Trennung von Stammdaten und Bewegungsdaten im BW

Alle Stammdaten im Business Information Warehouse sind als zeitabhängig deklarierbar. Dadurch erhalten Attribute, Texte oder Hierarchien die zusätzliche Information, von wann bis wann sie genau gelten.[5] Diese Zuordnung erfolgt vom System eindeutig während der Ladevorgänge zum Befüllen des Data Warehouse. Die Verknüpfung von Info-Cubes mit zeitabhängigen Stammdaten erfolgt über gesonderte SID-Tabellen, die ebenfalls der Verbindung von Bewegungs- und Stammdaten dienen. Neben diesen gibt es noch die reinen Stammdaten-Tabellen, in denen nicht die SIDs, sondern die konkreten Attributwerte gespeichert werden (siehe Abb. 6).

[5] Für Abfragen an Modelle mit zeitabhängigen Objekten ergibt sich die Besonderheit, dass in der Abfrage festgelegt werden muss, welcher Stichtag für die Selektion herangezogen werden soll. Dieser wird als *Query Key Date* bezeichnet und ist innerhalb einer Abfrage für alle zeitabhängigen Objekte gleichermaßen gültig.

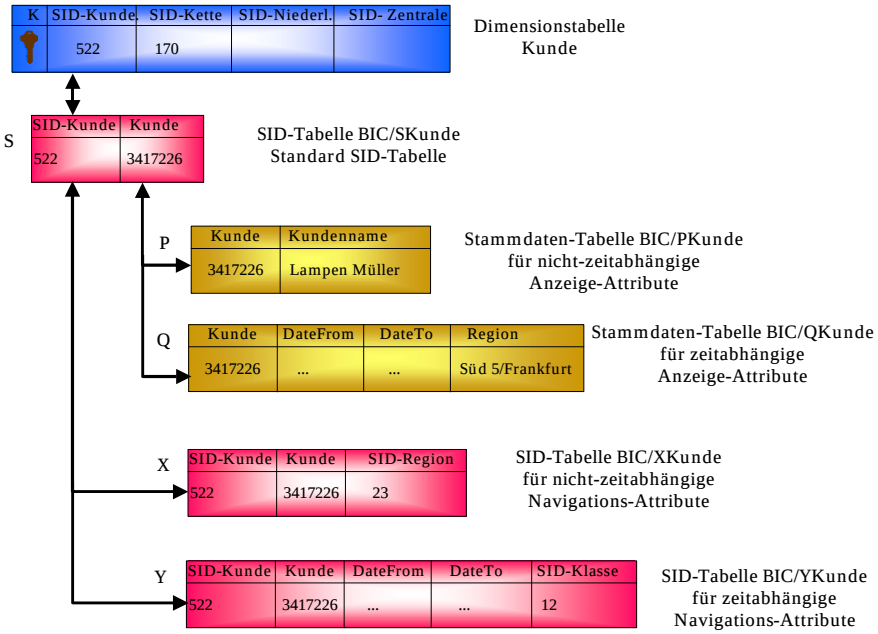


Abb. 6: Zusammenhang der verschiedenen Stammdaten-Tabellen

Der Zugriff auf das Modell bei konkreten Abfragen erfolgt immer von den Dimensionstabellen ausgehend, je nach konkreter Fragestellung dienen unterschiedliche Stammdaten-Tabellen zur näheren Eingrenzung und Bestimmung der qualifizierten Tupel aus der eigentlichen Dimensionstabelle, über die der Join mit der Faktentabelle durchgeführt wird.

2.4 Spezielle Eigenschaften des erweiterten Star Schemas

Eine wesentliche Metadaten-Information zu einem Info-Objekt als elementarstem Grundobjekt im Business Information Warehouse ist dessen Datentyp. Bei numerischen Datentypen gibt es neben den klassischen Formen die Möglichkeit, Mengen und Beträge zu definieren. Die Werte werden dann zusammen mit einer Einheit im System gespeichert, Geldbeträge haben dann zum Beispiel die Währung mit als Einheit.
Im Modell werden die Einheiten durch eine automatisch generierte Dimension abgebildet. Hierdurch ist das BW in der Lage, neben den Umrechnungen bei konstanten Verhältnissen, wie beispielsweise der Umrechnung von Mengenangaben mit Einheiten wie etwa Kilogramm und Tonnen, auch mit variablen Wechselkursen umgehen zu können. Die verschiedensten Umrechnungsvarianten sind durch selektierbare Kursumrechnungsarten darstellbar. Wird diese Funktionalität nicht benötigt, ist durch klassische numerische Datentypen der Overhead der Einheiten-Verwaltung vermeidbar und resultiert in schlankeren Modellen.

Die Einheiten-Umrechnung und Mehrwährungsfähigkeit sind wesentliche Leistungspotentiale des erweiterten Star Schemas, wie es im SAP BW zum Einsatz kommt.

3 Modellierungsvarianten hierarchischer Dimensionsstrukturen

In mehrdimensionalen Modellen definieren die abgebildeten Dimensionsstrukturen die Grundlage für analytische Operationen des On-Line Analytical Processing. Neben verschiedenen Aspekten der Darstellung statischer Dimensionsstrukturen spielt für die Berücksichtigung von Berichtsanforderungen insbesondere der Umgang mit Veränderungen innerhalb von hierarchischen Strukturen eine Rolle. Dieser Aspekt ist schon bei der Modellierung mit zu berücksichtigen.

3.1 Alternative Lösungsmöglichkeiten bei Strukturbrüchen

Der Zeitbezug ist für die Daten in mehrdimensionalen Modellen elementar, was sich auch in der besonderen Bedeutung der Zeitdimension ausdrückt, denn durch diese erfolgt inhärent die Historisierung der Fakten bzw. Bewegungsdaten. Die im Modellierungsprozess immer wieder aufkommende Frage nach dem Umgang mit strukturellen Veränderungen in Dimensionen ist für den Aufbau von Data Warehouse- und OLAP-Systemen entscheidend, da hierdurch die Möglichkeiten der Analysen festgelegt sind.
Eine häufig anzutreffende Berichtsanforderung ist die Analyse aller Daten nach der jeweiligen aktuellen Struktur, nach einer historischen Struktur oder nach der jeweils gültigen Struktur („historische Wahrheit“). Eine andere Anforderung an das Berichtswesen kann sein, nur die Strukturbestandteile zu berücksichtigen, die im gesamten Berichtszeitraum gültig sind („Äpfel nicht mit Birnen zu vergleichen“). Zur Abbildung dieser Berichtsanforderungen stehen die folgenden vier Möglichkeiten zur Verfügung:

1. Anpassung des historischen Datenmaterials an neue Strukturen
2. Separate Speicherung des historischen Datenbestandes zusätzlich zum Komplettbestand mit neuen Strukturen
3. Aufbau paralleler Hierarchien mit alten bzw. neuen Strukturen
4. Temporale Datenbanken und Gültigkeitsstempel

Die Anpassung des alten Datenmaterials an die neuen Strukturen hat den Vorteil, dass der Datenbestand nicht aufgebläht wird und die Datenstrukturen überschaubar bleiben. Da die alten Strukturen aber verloren sind, können nicht mehr alle Berichtsanforderungen abgedeckt werden, nur Berichte entsprechend der aktuellsten gültigen Struktur sind abrufbar. Dies ist die einfachste Form des Umgangs mit Strukturbrüchen in Dimensionen.
Die Berücksichtigung von alten und neuen Strukturen wird möglich, wenn eine getrennte Speicherung der Daten nach den jeweiligen Strukturen erfolgt. Bei dieser Vorgehensweise können alte Auswertungen abgerufen werden, dies impliziert aber ein größeres Datenvolumen und aufwendigere Verfahren der Aktualisierung. Durch den Aufbau paralleler Hierarchien sind nicht nur die alten Daten nach alten Strukturen abrufbar, vielmehr können alle Zahlen mit beliebigen Strukturen angezeigt werden. Die dadurch entstehenden Dimensionsstrukturen sind aber eher unübersichtlich und die Darstellung

neuer Zahlen nach alten Strukturen kann für Endbenutzer verwirrend sein. Die Historisierung über Zeitstempel wie in temporalen Datenbanken ermöglicht ein umfassendes Berichtswesen mit beliebigen historischen Varianten von Strukturen. Nachteilig sind aber die tendenziell schlechtere Performance und teilweise noch unausgereifte Konzepte.

3.2 Hierarchische Struktur zwischen Merkmalen innerhalb einer Dimension

Im Business Information Warehouse sind die Daten konsequent in die zwei Bereiche der Bewegungsdaten und der Stammdaten getrennt, die über jeweils unterschiedlichen Ladeprozesse in das BW transportiert werden. Die Bewegungsdaten beziehen sich immer auf einen Info-Cube, die geladenen Stammdaten stehen jedoch Info-Cube-übergreifend zur Verfügung.
Bei der Modellierung einer Dimensionshierarchie über Merkmale in einer Dimension wird jede Konsolidierungsebene der Hierarchie auf ein Merkmal abgebildet, die Hierarchie ist somit in den Bewegungsdaten abgebildet (siehe Abb. 7). Ein typisches Beispiel hierfür im BW ist eine Zeithierarchie mit den Merkmalen Tag, Monat und Jahr oder eine Hierarchie über die Merkmale Kunde, Region und Land. Die Merkmale in einer Dimension qualifizieren eine Wertinformation in einem Bewegungsdatensatz.

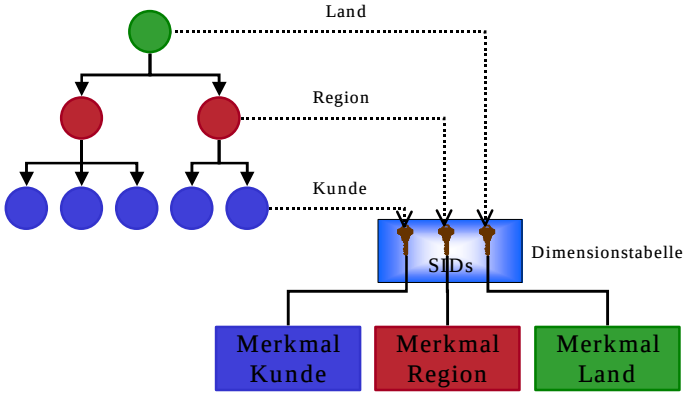


Abb. 7: Dimensionsstrukturen über Bewegungsdaten

Bei dieser Art der Hierarchiemodellierung werden nicht alle möglichen Strukturtypen unterstützt. Da jede Konsolidierungsstufe mit einem Merkmal korrespondiert, sind balancierte Strukturen eine Voraussetzung, die nur mit besonderer Kenntnis beim Anwender umgangen werden kann. Da die hierarchische Information in den Bewegungsdaten abgebildet ist und in den Dimensionen ein künstlicher Schlüssel die Merkmale einer Dimension zusammenfasst, ist es sogar möglich, die Daten auf einer Konsolidierungsebene statt für ein Basiselement zu laden. Diese Beziehung z. B. zwischen Kunde, Region und Land ist im BW technisch nicht gepflegt, alle Merkmale stehen gleichberechtigt nebeneinander, und es gibt daher auch keine vordefinierten Navigationspfade. Bei einem

drill können somit einzelne Ebenen auch übersprungen werden. Dies ist seit der Version 3.0 des Business Information Warehouse zumindest in der Darstellung im Business Explorer, dem Excel-basierten OLAP Analysewerkzeug, das mit dem BW ausgeliefert wird, anders darstellbar, da die Möglichkeit der Gruppierung von Merkmalen zu einer Anzeigehierarchie mittlerweile möglich ist.
Strukturelle Änderungen in der hierarchischen Beziehung zwischen Merkmalen sind bei dieser Abbildungsform ohne Datenreorganisation nicht abbildbar, das Berichtswesen kann nur auf Basis der tatsächlich gebuchten Zuordnung, d. h. der historischen Wahrheit, erfolgen.

3.3 Modellierung von Hierarchien über Navigationsattribute

Ein nicht unerheblicher Aspekt beim Aufbau eines Data Warehouse ist die Frage nach den Veränderungen in Dimensionshierarchien im Zeitablauf. Zum einen gibt es die Anforderung, verschiedene Strukturvarianten berücksichtigen zu können. Diese wird detailliert unter dem Stichwort *Zeitabhängigkeit* diskutiert. Zum anderen kann minimal gefordert werden, dass eine Reorganisation des Datenbestandes zur Anpassung an veränderte Strukturen erfolgt. Sind die einzelnen Konsolidierungsebenen Merkmale in den Bewegungsdaten, ist eine Reorganisation nur mit einem Neuaufbau des Info-Cubes abbildbar und daher sehr aufwendig. Das ist ein Grund, warum die Modellierung in den Stammdaten einen Vorteil darstellt. Dabei korrespondiert jede Konsolidierungsebene mit einem Attribut zu dem Merkmal der Basiselemente (siehe Abb. 8). Im Extremfall besteht eine Dimension dann nur aus einem einzigen Merkmal und alle hierarchischen Informationen sind in den Stammdaten dieses Merkmals abgebildet. In diesem Fall ist der Vorteil einer Line-Item-Dimension nutzbar, d. h. die Dimensionstabelle als Transfertabelle zwischen einem künstlichen Dimensionsschlüssel und den SIDs der Merkmale entfällt und die Anbindung an die Faktentabelle erfolgt direkt über die SID-Tabelle.
Die Ablage der Hierarchie-Information in den Stammdaten hat zur Konsequenz, dass diese allen Info-Cubes gleichermaßen zur Verfügung steht, eine Veränderung leicht möglich ist, da nur Stammdatenänderungen durchgeführt werden müssen, diese aber auch für alle Info-Cubes greifen. Eine Analyse auf Konsolidierungsebenen entsprechend dieser Navigationsattribute erfolgt dann nach der geänderten Struktur für alle Werte in den betroffenen Info-Cubes. Wie auch bei der Modellierung über Merkmale in der Dimension direkt bieten die Navigationsattribute keine vordefinierten Konsolidierungspfade, bei der Navigation können daher Ebenen übersprungen werden, der Anwender muss aber von den Attributen, die die Hierarchie formen, Kenntnis haben.

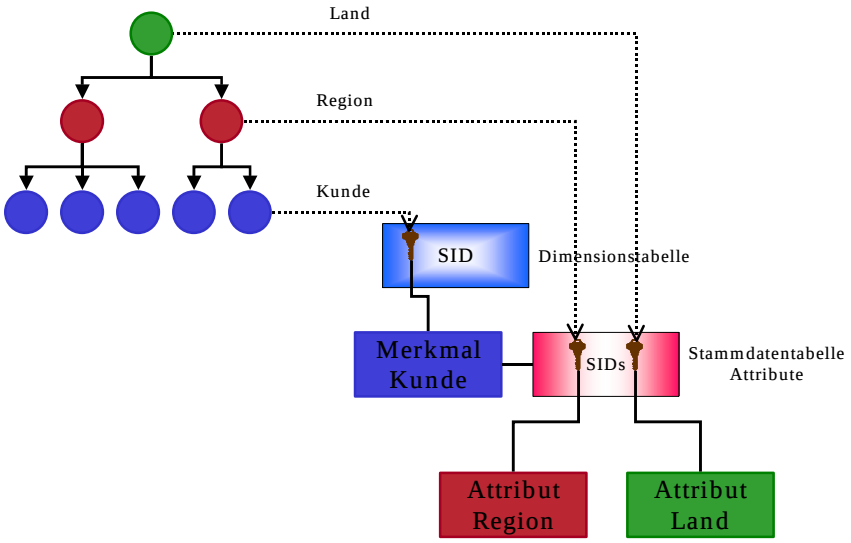


Abb. 8: Navigationsattribute als Grundlage für hierarchische Beziehungen

Mit der Modellierungsalternative der Hierarchieabbildung über Navigationsattribute steht in dieser Form ein Berichtswesen auf Basis der jeweils aktuellen Dimensionsstruktur zur Verfügung, eine Auswertung auf Basis historischer Zuordnungen kann erst mit einer zeitabhängigen Ausgestaltung der Navigationsattribute abgebildet werden. Bei zeitabhängigen Attributen ist zu beachten, dass die Möglichkeit der Bildung von Aggregaten zur Performancesteigerung auf diesen Attributen nicht mehr möglich ist.

3.4 Externe Hierarchien in den Stammdaten

Die bzgl. der Dimensionsstruktur flexibelste Art der Modellierung von hierarchischen Strukturen in Dimensionen sind im BW die Externen Hierarchien, da sie auf einer Darstellung in Form von rekursiven Beziehungen basiert [Sa00b]. Sie bietet sich daher insbesondere bei unbalancierten Dimensionsstrukturen an. Externe Hierarchien sind in den Stammdaten abgelegt und somit für alle Info-Cubes, die ein spezielles Merkmal verwenden, übergreifend gültig. Neben der Möglichkeit, verschiedene Hierarchien für ein Merkmal zu definieren, können einzelne Hierarchien zusätzlich in verschiedenen Versionen gepflegt sein.
Die Elemente Externer Hierarchien sind die Knoten, die ggf. Vorgänger und Nachfolger haben können. Elemente ohne Vorgänger sind Wurzelelemente, die ohne Nachfolger werden Blattelemente genannt. Die Bausteine einer Externen Hierarchie sind zum einen die Merkmalsknoten und zum anderen die Textknoten, wobei nur zu Merkmalsknoten Daten in das BW geladen werden können. Merkmalsknoten werden aus den Ausprägungen eines Info-Objektes aufgebaut, Textknoten sind frei definierbare Elemente einer Hierarchie. Für die Anzeige werden die Texte der Merkmalsknoten aus den zugrundelie-

genden Stammdaten des Merkmals gezogen, zu Textknoten wird der Anzeigetext direkt in der Hierarchiepflege gespeichert.
Die Externen Hierarchien eines Merkmals sind Bestandteil der Stammdaten und die Ablage erfolgt in mehreren Tabellen. Zentral ist die Tabelle, in der die rekursiven Beziehungen abgelegt werden. In dieser Inclusion Table genannten Tabelle werden die Beziehungen paarweise über Schlüssel abgelegt. Die Merkmalsknoten werden über den SID-Schlüssel des zugrundeliegenden Merkmals identifiziert, Textknoten bekommen einen identifizierenden Schlüssel mit negativem Vorzeichen, deren Texte in einer eigenen Tabelle gespeichert sind (siehe Abb. 9). Die Identifikation von Konsolidierungsebenen in dieser Form von Hierarchie erfolgt im BW nur auf Basis der Ebene, d. h. der Anzahl der Kanten ausgehend von der Wurzel bis zum Dimensionselement. Die Abbildung von Dimensionsstrukturen, in denen die Bedingung einer 1:n-Beziehung zwischen den einzelnen Konsolidierungsstufen fallen gelassen wird, ist in der Form möglich, dass ein Knoten zu mehreren übergeordneten Elementen zugeordnet werden kann und die Gewährleistung der konsistenten Verdichtung im BW automatisch erfolgt.

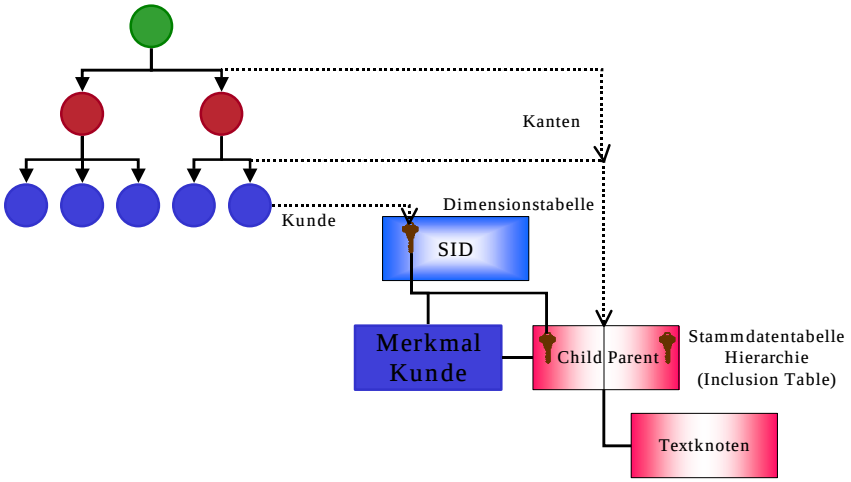


Abb. 9: Explizite Modellierung von Dimensionsstrukturen in Externen Hierarchien

Externe Hierarchien werden oft auch als Hierarchien im eigentlichen Sinne bezeichnet und über verschiedene Versionen von Hierarchien lassen sich schon Berichtsanforderungen abdecken, die unterschiedliche definierte Dimensionsstrukturen abbilden. Durch die zeitabhängige Modellierung von Hierarchien lässt sich dies auch direkt an einem Datum festmachen. Dieses Datum ist dann in einer Abfrage für alle zeitabhängigen Strukturbestandteile gültig.

3.5 Entscheidungshilfen zur Dimensionsmodellierung

Die dargestellten Varianten im Kontext der Modellierung hierarchischer Dimensionsstrukturen verdeutlichen die Komplexität des BW-Datenmodells. Als Hilfe zur Wahl der Modellierungsvariante im Rahmen des Modellierungsprozesses erfolgt nun eine Gegenüberstellung der Alternativen anhand einzelner Kriterien.

Als eine erste elementare Entscheidungshilfe kann die Form der zu berücksichtigenden *Historisierung* angeführt werden. Nur in der Abbildung über Merkmale in den Bewegungsdaten kann die historische Sicht zum Buchungszeitpunkt implementiert werden. Die aktuell gültigen Strukturen sind bei Navigationsattributen und externen Hierarchien der Standard, wobei in beiden Fällen durch den Übergang zu zeitabhängigen Strukturen eine Historisierung ermöglicht wird. Bei externen Hierarchien ist es darüber hinaus auch möglich, einen zeitlichen Bezug über verschiedene Versionen oder Namen abzubilden.

Das Stammdatenkonzept des BW führt zu einem zweiten Anhaltspunkt, an dem sich die Wahl der Modellierungsform orientieren kann. Der *Wirkungsbereich* von Merkmalen in Bewegungsdaten ist gerade der Info-Cube, die damit abgebildete Struktur ist so zunächst nicht in anderen Info-Cubes verfügbar. Demgegenüber sind externe Hierarchien wie auch Navigationsattribute Bestandteil der Stammdaten und damit für alle Info-Cubes eines Systems gültig.

Unabhängig vom Gesichtspunkt der Transformation eines semantischen Datenmodells in das BW-Modell gibt es immer Aspekte des physischen Datenmodells, die einen signifikanten Einfluss auf die Abbildung der Datenstrukturen haben. Im SAP BW spielt die *Performance* eine herausragende Bedeutung, da diese einen kritischen Faktor darstellt. Dieser legt den Verzicht auf zeitabhängige Strukturen ebenso nahe, wie den sparsamen Einsatz von Navigationsattributen. Auf externe Hierarchien kann oftmals ohne Aggregate gar nicht performant zugegriffen werden.

Das Konzept der mehrdimensionalen Datenmodellierung impliziert eine Vorstellung von den *Navigationspfaden* innerhalb der Dimensionshierarchien. Nur in externen Hierarchien sind diese direkt berücksichtigt, in der Darstellung von hierarchischen Strukturen über Merkmale oder Navigationsattribute ist dieser Zusammenhang nicht modelliert, sondern muss sich aus der Kenntnis des Modells ergeben.

Die möglichen abbildbaren *Dimensionsstrukturen* bilden ein weiteres Entscheidungskriterium, denn nur externe Hierarchien bieten die volle Flexibilität. Da Merkmale und Navigationsattribute jeweils mit einer festen Ebene in der Dimensionsstruktur korrespondieren, sind damit im allgemeinen nur balancierte Strukturen darstellbar.

Dieses Kriterium erfährt noch eine Verschärfung durch die Anforderung, auch *m:n-Beziehungen* in Dimensionen zu unterstützen. Externe Hierarchien erlauben Blätter mit mehreren übergeordneten Elementen, in Bewegungsdaten ist dies nur in der Form, wie es gebucht wurde, möglich. Über Navigationsattribute ist dies nicht implementierbar.

Diese angeführten Kriterien beziehen sich auf die statischen Aspekte, aber die Anforderungen im Rahmen des ETL-Prozesses zur Befüllung des Data Warehouse resultieren in einem weiteren Anhaltspunkt zur Entscheidungshilfe. Die performante *Reorganisation* von Datenbeständen bei Strukturveränderungen ist insbesondere bei großen Installationen, in denen nur einen kleines Ladefenster für die ETL-Prozesse verbleibt, sehr wichtig. Bei Bewegungsdaten-basierten Strukturen ist dies nur mit einem kompletten Neuaufbau möglich. Externe Hierarchien lassen sich hingegen sehr schnell flexibel verändern.

4 Zusammenfassung

Analyseorientierte Informationssysteme, deren Fokus in der zeitnahen Versorgung betrieblicher Entscheidungsträger mit relevanten Informationen zu Analysezwecken liegt, zielen auf die Unterstützung der dispositiven und strategischen Prozesse in Unternehmen ab. Genau diesen Bereich adressiert das Business Information Warehouse als zentrale Data Warehouse-Lösung und bildet damit die Grundlage für vielfältige analytische Anwendungen. Wichtige Eigenschaften des Systems der SAP sind dabei die Möglichkeiten der Währungsumrechnung und das Stammdatenkonzept. Von anderen Lösungen hebt es insbesondere der mitgelieferte Business Content ab, denn dieser bietet gerade für R/3-Quellsysteme vorgefertigte betriebswirtschaftliche Schablonen zum schnellen Aufbau von analyseorientierten Applikationen auf Basis einer Datenhaltung im Business Information Warehouse.

Online Analytical Processing (OLAP) als Grundprinzip für den Aufbau von Systemen zur Unterstützung von Fach- und Führungskräften in ihren analytisch geprägten Aufgaben basiert im Kern auf einer mehrdimensionalen konzeptionellen Sicht auf die Daten mit Möglichkeiten der Navigation in den Würfeln mit beliebigen Projektionen und auf verschiedenen Verdichtungsstufen. Die Möglichkeiten zur Abbildung vielfältiger Strukturen für Konsolidierungspfade sind ein Aspekt, der die Analysemöglichkeiten stark beeinflusst. Hierbei spielen auch Aspekte der Zeitabhängigkeit und der Umgang mit Änderungen in Dimensionen eine Rolle.

Der Modellierung hierarchischer Dimensionsstrukturen kommt im Kontext analyseorientierter Informationssysteme eine besondere Bedeutung zu, daher unterstützt das BW vielfältige Dimensionsstrukturen.

Bei der Abbildung von Hierarchien im Business Information Warehouse gibt es die drei grundsätzlichen Möglichkeiten der Darstellung über Merkmale in einer Dimension, über Navigationsattribute des Basismerkmals sowie über Externe Hierarchien. Dies ist ein Freiheitsgrad der Datenmodellierung. Für die Entscheidung, welche der Varianten in einer konkreten Anwendung vorzuziehen ist, sind verschiedene Kriterien heranzuziehen. Erstes Unterscheidungsmerkmal ist der Wirkungsbereich, denn der Ansatz des unternehmensweiten Data Warehouse mit einem zentralen konsistenten Modell propagiert die Modellierung in Stammdaten, die für alle Info-Cubes gültig sind. Ein sehr bedeutendes Kriterium ist die Performance, wobei vom Grundprinzip her in Dimensionen direkt modellierte hierarchische Beziehungen tendenziell performanter sind als Navigationsattribute und Externe Hierarchien. Die Kriterien der flexiblen Reorganisation sowie der Vielfalt an unterstützten Dimensionsstrukturtypen sprechen in vielen Fällen für die Modellierung über Externe Hierarchien. Die drei Varianten schließen sich aber nicht gegenseitig aus, auch die gleichzeitige Abbildung auf verschiedenen Wegen ist im Modell möglich. Ein weiteres wesentliches Kriterium berücksichtigt die Anforderungen an die Unterstützung für Dimensionsstrukturänderungen und deren unterschiedlichen Anforderungen an ein Berichtswesen. Die Historisierungen und Nachverfolgung von Strukturänderungen werden unter dem Stichwort Zeitabhängigkeit diskutiert. Hier bietet das Business Information Warehouse Möglichkeiten, über die zeitabhängige Ausgestaltung von Stammdaten verschiedene Berichtsszenarien abzubilden. Die hinzukommende Flexibilität in den Berichtsanforderungen wird allerdings mit einer erhöhten Komplexität und schlechterer Performance erkauft.

Literaturverzeichnis

[Gl97] Gluchowski, P.: Data Warehouse-Datenmodellierung – Weg von der starren Normalform. In: Datenbank-Fokus, Nr 11, 1997; S. 62-66.

[Ha99] Hahne, M.: Logische Modellierung für das Data Warehouse – Bestandteile und Varianten des Star Schemas. In (Chamoni, P.; Gluchowski, P. Hrsg.): Analytische Informationssysteme – Data Warehouse, On-Line Analytical Processing, Data Mining. Springer-Verlag, Berlin, 1999; S. 145-170.

[Ha02] Hahne, M.: Logische Modellierung mehrdimensionaler Datenbanksysteme. Deutscher Universitäts-Verlag, Wiesbaden, 2002.

[Nu98a] Nußdorfer, R.: Star Schema, das E/R-Modell steht Kopf – Teil 1: Modellieren von Faktentabellen. In: Datenbank-Fokus, Nr 10, 1998; S. 22-28.

[Nu98b] Nußdorfer, R.: Star Schema – Teil 1: Modellierung von Dimensionstabellen. In: Datenbank-Fokus, Nr 11, 1998; S. 16-23.

[Sa00a] SAP: Multidimensional Modeling with BW. ASAP for BW Accelerator, 2000.

[Sa00b] SAP: Hierarchies in SAP BW. ASAP for BW Accelerator, 2000.

Transbase®: A leading-edge ROLAP Engine supporting multidimensional Indexing and Hierarchy Clustering[∇]

R. Pieringer[1], K. Elhardt[1], F. Ramsak[2], V. Markl[3], R. Fenk[2], R. Bayer[2]

[1] Transaction Software GmbH
Thomas-Dehler-Str. 18
D-81737 München
{pieringer,elhardt}@transaction.de

[2] Bayerisches Forschungszentrum für Wissensbasierte Systeme
Boltzmannstr. 3
D-85747 München
{ramsak,fenk}@forwiss.de, bayer@in.tum.de

[3]IBM Almaden Research Center, K55/B1,
650 Harry Road, San Jose, CA 95120-6099
marklv@us.ibm.com

Abstract: Analysis-oriented database applications, such as data warehousing or customer relationship management, play a crucial role in the database area. In general, the multidimensional data model is used in these applications, realized as star or snow-flake schemata in the relational world. The so-called star queries are the prevalent type of queries on such schemata. All database vendors have extended their products to support star queries efficiently. However, mostly reporting queries benefit from the optimizations, like pre-aggregation, while ad-hoc queries usually lack efficient support. We present the DBMS Transbase® in this paper, which provides a new physical organization of the data based on hierarchical clustering and multidimensional clustering combined with multidimensional indexing. In combination with new query optimizations (e.g., hierarchical pre-grouping) significant performance improvements are achieved. The paper describes how the new technology is implemented in the Transbase® product and how it is made available to the user as transparently as possible. The benefits are illustrated with a real-world data warehousing scenario.

1 Introduction

Data warehousing (DW), online analytical processing (OLAP), and customer relationship management (CRM), have become a major market in the database area through the last decade. The multidimensional paradigm seems to be the undisputed winner as a design choice for such databases. The conceptual model adopted is a data warehouse consisting of facts (or measures) organized into a set of dimensions, which in turn are organized into levels of different aggregation granularity (i.e., detail) that comprise one or more hierarchies. Even though proprietary multidimensional database management systems (DBMSs) exist, the vast majority of systems use relational DBMSs as the underlying storage system.

For relational databases, the multidimensional data warehouse consists of one or more *star schemata* [CD97a], featuring a central fact table

[∇] This work is funded by the European Union under the IST-Project EDITH (IST-1999-20722)

surrounded by so-called dimension tables. The most prevalent kind of queries submitted to such a system is the *star query*. Star queries impose restrictions on the dimension values that are used for selecting specific facts; these facts are further grouped and aggregated according to the user demands. The join of the central (and usually very large) fact table with the surrounding dimension tables (also known as a *star join*) has been identified as frequent, major bottleneck in evaluating such queries. Various solutions have been proposed over the years to cope with these problems. Indexing schemes [NG95, NQ97, Sar97, CI98, WB98, Wu99, WOS01] and precomputation of aggregation results [GM95, Rou98, Sri96] have been studied extensively in the research community and are also, to some extent, used in commercial systems [ACN01, Ora01, Zah00].

While these solutions work well in reporting scenarios, they do not support acceptable performance for *ad hoc* star queries, i.e., queries that are not known in advance, which become more and more important in online applications. For this kind of queries the usage of precomputed aggregation results is extremely limited or even impossible in some cases. Even when elaborate indexes are used, due to the arbitrary ordering of the fact table tuples, there might be as many I/Os as there are tuples in the result set.

To overcome these deficiencies, new alternatives of the physical organization of data have emerged [Des98, MRB99, KS01]. The idea is to incorporate the two fundamental properties of the conceptual data model into the data storage: the multidimensionality and hierarchy semantics. These organizations exploit a special kind of key that is based on the hierarchy paths of the dimensions, in order to achieve hierarchical clustering of the facts. This physical clustering results in a reduced I/O cost for the majority of star queries, which are based on the dimension hierarchies. Moreover, [MRB99] and [KS01] exploit a clustering, multidimensional index for storing the tuples. A typical star join then is transformed into a multidimensional range query, which is very efficiently computed using the underlying multidimensional data structures.

In this paper we present the Transbase® DBMS from Transaction Software GmbH that incorporates state of the art techniques for analysis-centric applications. More precisely, it supports the UB-Tree as native multidimensional index and allows for clustering of data according to hierarchy semantics. Taking advantage from the knowledge of hierarchies, not only the physical storage of the data can be optimized but also query processing can largely be improved. We will discuss the basic concepts of the underlying technology as well as how it is implemented in the DBMS kernel. The evaluation in a real-world data warehousing scenario shows significant performance improvements over traditional techniques.

The rest of the paper is organized as follows. Section 2 covers related work and in Section 3 we introduce the basic technology used in Transbase®. Section 4 shows the user's view with an example before we address the implementation issues in Section 5. Section 6 covers the automatic maintenance of hierarchy clustering and Section 7 is dedicated to the query processing for multidimensional hierarchical clustering. In Section 7 we provide an evaluation of the techniques in a real-world scenario; Section 9 summarizes our contribution.

2 Related Work

Due to the large body of work in the area of data warehousing and optimizing DBMSs for these applications we can only give an incomplete account of related work in this field. We cover general approaches first and then address commercial solutions.

2.1 Star query optimization

One of the most important parts of a star query is the processing of the star join. Star join processing has been studied extensively and specific solutions have been also implemented in commercial products. See [CD97b] for an overview.

To compute the star join, most systems avoid building the Cartesian product of the fact table with the dimension tables as the resulting cardinality leads to a non-tolerable overhead. Thus, one tries to apply dimension restrictions also to the fact table in order to reduce the join size. Bitmap indices are often used to speed up the access to the fact table. The bitmaps corresponding to the different dimension values are ANDed or ORed depending on the selection condition. The resulting bitmap is used to extract tuples from the fact table [NG95, NQ97]. When the query selectivity is high, only a few bits in the result bitmap are set. If there is no particular order (clustering) among the fact table tuples, we can expect each bit to access a tuple in a different page.

Multidimensional clustering has been discussed in the field of multidimensional access methods (e.g., [GG97] and [Sam90]). [ZSL98] addresses the issue of hierarchical clustering for the one-dimensional case. The importance of good physical clustering in OLAP has been shown in [KR98], where packed R-trees are exploited for storing the results of the data cube operator ([Gra96]). In [Des98], the benefits of hierarchical clustering for star queries was observed as a side effect of using a chunked file organization for enabling caching with chunk as the caching unit.

Among others, in [MRB99] the UB-tree (see Section 3.1) is used as a primary organization of the fact table. Surrogate keys based on the dimension hierarchies are exploited and hierarchical clustering of the fact table is achieved. Consequently star joins are transformed to multidimensional range queries. The combination of the two mechanisms results in a greatly reduced I/O cost for star joins.

In [KS01] a physical organization based on a hierarchical chunking of the fact table is presented. Fact data are clustered physically according to the dimensional hierarchies. To achieve this clustering, special path-based dimension keys are exploited. In particular, these keys guide the clustering (called *chunking*) process. Star joins are transformed to range queries in the multidimensional and multi-level data space of a cube. The adopted multidimensional structure is a variant of the Grid File [NHS84].

Several aspects of processing and optimizing star join queries on hierarchically clustered fact tables are also presented in [TT01]. The paper considers a star schema with UB-Tree organized fact tables and dimension tables stored sorted on a composite surrogate key. For a particular class of star join queries, the authors investigate the usage of sort-merge joins and a set of other heuristic optimizations.

A general query processing framework that addresses all issues involved in star query processing over hierarchically clustered fact tables has been presented in [Kar02] and [Pie03]. In this paper, we describe in more detail, how this framework is implemented in a real DBMS. We extend the concepts of pre-grouping, first introduced in [CS94], [YL94] and [YL95] by including hierarchical information.

2.2 Hierarchies in Oracle

Hierarchies in Oracle are not used, in order to cluster data in the fact table ([Ora01]). Hierarchies are additional information (meta data) for special features for query processing. For example, query rewriting needs hierarchical information to use materialized pre-aggregated views in the query formulation. Thus, the definition of dimensions and hierarchies does not influence the physical clustering of the data.

A dimension, created by a `CREATE DIMENSION` statement contains one or more hierarchies. The hierarchy levels can be placed in any table. For each hierarchy level, one or more feature attributes can be assigned (via the `DETERMINES` clause). Hierarchies can be modified by adding or dropping hierarchy levels.

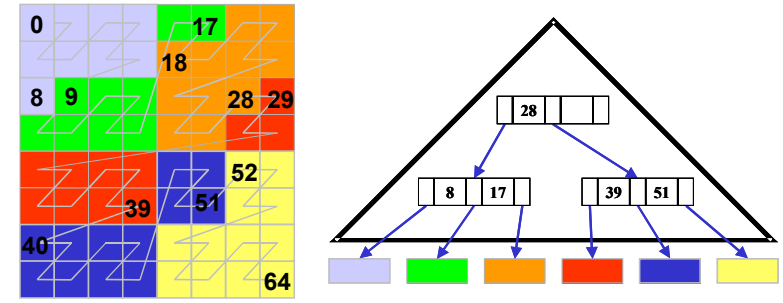
3 Basic Concepts

In this section we briefly introduce the fundamental concepts that are used in the Transbase® DBMS to support OLAP applications. On the one hand, we introduce the multidimensional index available in Transbase® and on the other hand explain the basic idea of hierarchy clustering.

3.1 The UB-Tree

We just give a short introduction to UB-Trees here, details can be found in [Bay97, Ram00]. The basic idea of the UB-Tree is to use a space-filling curve to map a multidimensional universe to one-dimensional space. Using the Z-curve for preserving multidimensional clustering it is a variant of the zkd-B-Tree [OM84]. A Z-address $\alpha = Z(x)$ is the ordinal number of the key attributes of a tuple x on the Z-curve, which can be efficiently computed by bit-interleaving. A standard B-Tree is used to index the tuples taking the Z-addresses of the tuples as keys. The pagination of the B-Tree creates a disjunctive partitioning of the multidimensional space into so-called Z-regions. This allows for very efficient processing of multidimensional range queries.

Figure 1 shows a Z-region partitioning for a two-dimensional universe and the corresponding B-Tree. The interval limits of the Z-regions are also depicted.



Z-region partitioning of 2-D space **B+-Tree with Z-values**

Figure 1 UB-Tree: Z-region partitioning and underlying B-Tree

The processing of basic operations, i.e., insertion, deletion, update, and point query, of the UB-Tree are analogous to the basic operations of the B-Tree. For each tuple the corresponding Z-address is computed, and with the resulting value the underlying B-Tree is accessed. Thus, all basic operations require only cost proportional to the height of the tree. The only recommendable modification to the standard B-Tree algorithms is an adaptation of the split algorithm to achieve a "good" (as rectangular as possible) Z-region partitioning.

A UB-Tree is especially good in processing multidimensional range queries, as it only retrieves all Z-regions that properly intersect the query box. Consequently, it usually shows the nice property that the response time of the range query processing is proportional to the result set size.

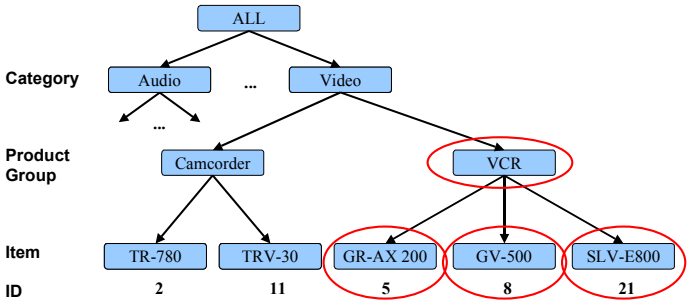
3.2 Clustering of Hierarchies

Hierarchies play an important role in various application domains. They are used to provide a semantic structure to data, e.g., a geographical classification of customers in a data warehouse. As the hierarchies cover the application semantics they are used frequently by users to specify the restrictions on the data as well as the level of aggregation. The restrictions on the hierarchies usually result in point or range restrictions[1] on some hierarchy levels [Sar97]. The problem that arises is that these restrictions on upper hierarchy levels lead to a large set of point restrictions on the lowest level, i.e., the level with the most detailed information. This situation is depicted in Figure 2 (a): restricting the level 'Product Group' to the value 'VCR' leads to the set of ids {5,8,21} and not to the interval [5,21], as the item with id 11 does not belong to the specified product group.

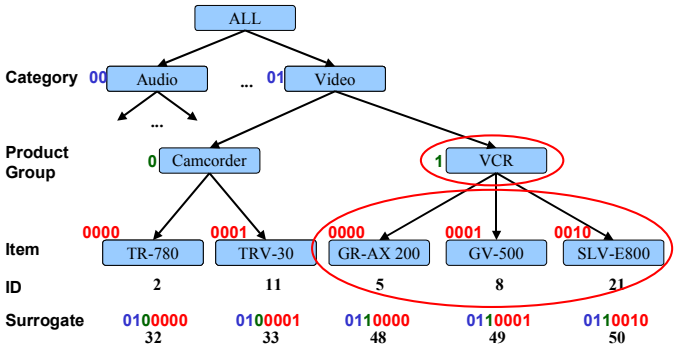
For most access methods it would be more efficient to process one range restriction instead of a set of point restrictions. The resulting question is how to map a point/range restriction on a higher hierarchy level to a range restriction on the lowest level? To this end, Transbase® applies the clustering scheme for hierarchies as proposed in [MRB99]. A special kind of keys is used for the elements of the lowest level which reflect the hierarchy semantics, i.e., keys which adhere to the partial order defined by the hierarchy levels. These so-called *(compound) surrogates* guarantee

[1] Range restrictions on hierarchy levels are only meaningful if an order on the elements of the hierarchy level is defined.

that the keys of all elements in a sub-tree of the hierarchy lie within a closed interval (Figure 2 (b)) such that a key of an element not lying in the subtree is not within the interval. In our example, the restriction to the product group 'VCR' now leads to the interval [48,50]; the item with id 11 is mapped to the surrogate 33 that does not violate the interval.



(a)



(b)

Figure 2 Example of hierarchy clustering: (a) non-clustered vs. (b) clustered hierarchy

We refer to this technique as *hierarchy clustering (HC)* from now on. If we combine HC and multidimensional indexing on multiple hierarchy encoding as it is done in Transbase®, then we speak of *multidimensional hierarchical clustering (MHC)*.

4 Example: MHC in Transbase®

In this section, we will briefly present the user's perspective when using the OLAP functionality in Transbase® ([Tra02]).

For our discussions we use a conventional star schema [CD97a] with a fact table consisting of *dimension* (qualitative) and *measure* (quantitative) attributes [Kim96]. For the dimensions typically one or more hierarchical classifications based on the dimension attributes (often referred to as *features*) exist. The primary key of the dimension represents the most detailed level of the dimension hierarchies.

In this paper, we focus on star schemata for the ease of description. However, the algorithms also are implemented for general snowflake schemata.

4.1 Sample Schema

As running example throughout this paper we use the schema depicted in Figure 3. This data warehouse stores sales transactions recorded per item, store, customer, and date. It contains one fact table *FACT*, which is defined over the dimensions: *PRODUCT*, *CUSTOMER*, *DATE,* and *LOCATION* with the obvious meanings. The measures of *FACT* are *price, quantity* and *sales* representing the values for an item bought by a customer at a store at a specific day. The schema of the fact and dimension tables is shown in Figure 3 and the dimension hierarchies are depicted in Figure 4.

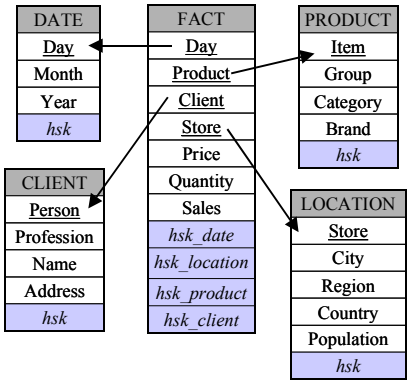


Figure 3: Sample Schema: standard star schema and HC extension (*hsk**)

The dimension *DATE* is organized in three levels: day – month – year. The dimension *CUSTOMER* is organized in two levels: customer – profession. For each customer the dimension table contains an ID, a name, an address, and a profession. The dimension has two hierarchical attributes (*person_id, profession*) and two feature attributes (*name, address*). The *LOCATION* dimension is organized by four levels: store – city – region – country. Stores are grouped into cities, these are grouped into regions and the regions finally are grouped into countries. For each city, the population is stored as feature attribute. The dimension has four hierarchical attributes (*store, city, region, country*) and one feature attribute (*population*) that is assigned to the city level.

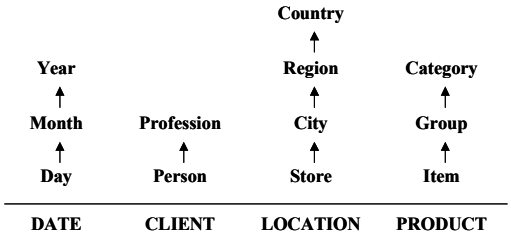


Figure 4: The dimension hierarchies of the example

Finally, the *PRODUCT* dimension is organized into three levels: item – group – category. Items are grouped into product groups and those are further grouped into categories (e.g., "air condition"). The attribute *brand* characterizing each item is a feature attribute.

Star queries are written in standard SQL, i.e., the join attributes between the fact and dimension tables are the dimension key attributes:

```
SELECT SUM(sales)
FROM FACT F, DATE D, PRODUCT P , LOCATION L
WHERE D.day=F.day AND P.item=F.product AND
L.store=F.store AND D.year=2002 AND P.category = 'Air
Condition' AND L.population > 1000000
GROUP BY D.year, D.month
```

This query returns the sum of sales for the year 2002 for air conditions in cities with a population larger than one million.

4.2 The User's View: Hierarchy Specification by extended DDL

To allow the user for defining such a schema, the DDL has been extended to express hierarchies and the desired physical organization of the fact table according to the dimension hierarchies. This is achieved by specifying an additional field per dimension table per hierarchy that basically represents the compound surrogate derived by HC as described earlier.

The keyword `SURROGATE` is used to mark the definition of a surrogate field as opposed to the definition of a standard user visible field. In the dimension table, we denote a compound surrogate by the keyword `COMPOUND`. The hierarchy levels are specified after this keyword by enumerating the hierarchy levels from top to bottom. The maximum fan-out of each hierarchy level is denoted by the keyword `SIBLINGS` (to specify the number of bits reserved for the hierarchy level) after the corresponding hierarchy level.

Figure 5 shows the DDL for the Location dimension: the surrogate specification defines the hierarchy depicted in the Figure 4. The `SIBLINGS` information specify that in this hierarchy there are at most 10 countries, at most 50 regions per country, at most 50 cities per region, and at most 1000 stores per city. Thus, the Location dimension may at most contain 10*50*50*1000=25.000.000 members.

```
create table Location (
       country char(*) NOT NULL,
       region char(*) NOT NULL,
       city char(*) NOT NULL,
       store char(*) NOT NULL,
       SURROGATE cs_location COMPOUND (
       country SIBLINGS 10, region SIBLINGS 50,
       city SIBLINGS 50, store SIBLINGS 1000),
       PRIMARY KEY (store)
);
```

Figure 5 Extended Create-Statement for dimension tables

For the fact table specification (see Figure 6), we now have to specify the physical organization, besides the standard relationships to the dimension tables. As we want to cluster the data according to the hierarchies of the

dimension, we use the surrogates instead of the "logical" keys. To this end, we introduce the concept of reference surrogates. Technically, a reference surrogate is an additional system maintained field in the fact table. Because of the necessary foreign key constraints in the fact table, it is possible to decide to which dimension the reference surrogate belongs.

We use again the keyword `SURROGATE` to denote a surrogate. Then we use the keyword `FOR`, in order to assign the reference surrogate to a dimension:

```
create table fact (
   product char(*) NOT NULL references product (item),
   store char(*)NOT NULL references location(store)
   time integer NOT NULL references date(day),
   sales numeric(10,3),
   price numeric(10,3),
   quantity numeric(10,3),
   SURROGATE cs_prod FOR product,
   SURROGATE cs_store FOR store,
   SURROGATE cs_time FOR time,
   PRIMARY HCKEY (cs_prod, cs_store, cs_time))
```

Figure 6 Extended Create-Statement for the fact table

A different keyword for the key specification the UB-Tree (`PRIMARY HCKEY`) is used to specify the index access method, namely a UB-Tree (HC stands for HyperCube as the UB-Tree is called in Transbase®).

All further statements (especially `INSERT` and `UPDATE`) may (and must) ignore the additional fields. This is comparable to the creation of a secondary index which is made up by the user but then becomes a system maintained part of the database.

It is important to note that all fields created by the `SURROGATE` specification are system maintained and are not visible to the user. Even though the fields are really stored in the tables, the user only works on a view of the table which projects the surrogate fields out.

As the example shows, the user can very naturally specify the physical organization of the data according to the schema semantics. In the following sections we will discuss the internals of the system transparent to the user.

5 Implementing HC in the DBMS Kernel

For the implementation of HC in the Transbase® kernel various issues have to be solved. The most important one is the internal representation and management of the surrogates as well as schema extensions, necessary for the automatic processing.

Before we continue, we want to introduce some terminology that is frequently used in the remainder of the paper. We refer to the fact table as *FT* and to the dimension tables as D_i. We use *FT.m* to denote a measure attribute of the fact table, $D_i.h_j$ to denote a hierarchical attribute (h_1 denotes the leave level of the hierarchy, i.e., it is the key of the dimension table), and $D_i.f_k$ denotes a feature attribute of the dimension. *PRED* and *AGGR* are placeholders for any predicate resp. aggregation function on the specified attribute.

5.1 Internal Representation of Compound Surrogates

As already indicated in Figure 3, the `SURROGATE` specification in the "create table" statement leads to the creation of an extra, non-visible, field in the dimension or fact table containing the encoding (*surrogate key, "hsk"*) of the corresponding hierarchy. It is unique for each dimension tuple as each leaf member of the hierarchy is assigned a unique value by HC.

The *hsks* are also contained in the fact table, as there they are required for the physical clustering of the data. As the logical dimension keys and the *hsks* are substitutes to each other, one may save a lot of space in the fact table if one would suppress the logical keys and just keep the *hsks*, if available for a dimension. However, this may lead to drastic performance decrease in cases where the original keys of the fact table tuples are accessed in the query. This would lead to expensive joins with the dimension tables. The suppression of logical keys is not implemented in Transbase®.

Internally, compound surrogates are fixed length bit strings. The length corresponds to the siblings specification of all surrogate components.

5.2 System Catalog Extension

In order to implement hierarchical clustering the system needs to have knowledge about the defined surrogates. The system catalog extension is designed to even allow several compound surrogates (and thus hierarchies) within one table.

5.3 Automatic Index Creation

For efficient query retrieval and surrogate maintenance, we need two special secondary indexes on the dimension table. Let *cs* be the field name of the compound surrogate and $h_t,..,h_1$ be the list of field names of the levels for the compound surrogate definition. Transbase® automatically creates the following indexes (here described by the following virtual SQL `CREATE` statements):

```
CREATE INDEX "@@sys_surrCSX_<surrid>" ON <dim_table>
(cs);
CREATE INDEX "@@sys_surrHX_<surrid>" ON <dim_table>
(h_t,..,h_1,cs);
```

Thus, the index names consist of two components, a prefix that marks the index as system index, and a generated suffix of the kind of the index (*CSX* or *HX*) and the surrogate id of the corresponding surrogate. The indexes are needed for the computation of compound surrogates, for the lookup of reference surrogates (see section 6.2) and for query processing. Of course, these indexes cannot be dropped by the user.

5.4 Multiple Hierarchies

A dimension of a data warehouse may include several independent hierarchies. Because of the representation of hierarchies by compound surrogates, we have to deal with several compound surrogates, one for every possible hierarchy.

An example of several hierarchies is shown in Figure 7. The customer dimension has a geographical hierarchy with country – region – town – customer and an organizational hierarchy with profession – customer.

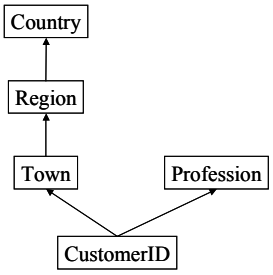


Figure 7: Customer Dimension with two Hierarchies

We have to distinguish between two levels of hierarchies. The conceptual level defines hierarchies on the conceptual data warehouse schema. Depending on the data warehouse application, multiple hierarchies may be defined on all dimensions representing the application data model. Usually hierarchies represent drill and aggregation paths for user queries.

The physical level of hierarchies is responsible for the clustering properties of the hierarchies, in combination with hierarchies of other dimensions, i.e., the complete MHC schema. The number of clustering hierarchies is restricted due to the properties of the clustering multidimensional index. It usually makes sense to use only one hierarchy per dimension for clustering, because in most times, hierarchies are dependent or one hierarchy is more important for user queries. However, in some cases, two or more hierarchies may be required for clustering (e.g., most user queries restrict both of these hierarchies). In addition, one dimension table may be used for several fact tables, that use different hierarchies for clustering. Thus, we have to provide multiple hierarchies for one dimension.

The internal structures allow to establish an arbitrary number of hierarchies, represented by compound surrogates. However, we require that the leaf level of all hierarchies is the same (a so called shared leaf level), usually the primary key of the dimension table. This hierarchy property is checked when creating the table and the compound surrogates in the DDL statement. With multiple hierarchies allowed on one dimension table, we can use the dimension table for several fact tables that can be clustered w.r.t. different hierarchies. So we avoid redundancy problems for replicated dimension tables.

Every compound surrogate is assigned a unique id. This so called *surrid* is referred to by the reference surrogates. One fact table includes an arbitrary number of reference surrogates specified by the *surrid* of the corresponding compound surrogate of the dimension tables. Thus, we can use reference surrogates of several hierarchies of one dimension table within one fact table. These reference surrogates may be used as index key attributes and thus for clustering the fact table according to several hierarchies.

6 Maintenance of MHC

After discussing the internal representation of the surrogates, we now turn to the automatic maintenance of the hierarchical clustering. We start with the insertion of new dimension members and continue with the insertion of fact table tuples. Finally, we will address the issue of major reorganization of dimension hierarchies.

6.1 Computation of Compound Surrogates

Computation of a compound surrogate *hsk* occurs when a tuple is inserted into the dimension table. Updates of hierarchy fields also may lead to a re-computation of *hsk*.

In the following picture, the insertion algorithm is depicted for our example product hierarchy. Figure 8 schematically shows the insertion of a new tuple. We assume the values v_i for each hierarchical field h_i. Considering the hierarchy, the insertion of a tuple means the creation of a new path $(v_k,\ldots,v_1)$ in the dimension D.

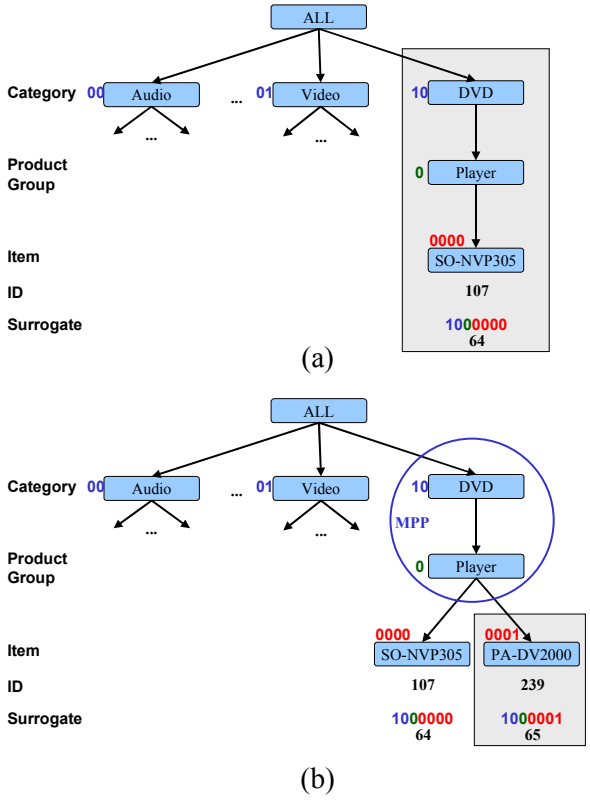


Figure 8: Insertion of new tuples into a dimension

For the computation of the compound surrogate *hsk* we have to check if there exists already a prefix of the new path in *D*. For a tuple to be inserted, we call the already existing part of the new path the *matching prefix path* (MPP). The MPP may be empty as in Figure 8 (a) – in this case a new root element, here "DVD", has to be created and the forest grows by one tree.

A non-empty MPP (see Figure 8 (b)) comprises levels (h_1,...,h_k) for some k with $k > 1$ and $k \leq t$. The number k then is called the *match level* of the new tuple's path (2 in our example). At the next lower level, i.e., the first non-matching level, the surrogate for the new value is determined. Usually, the maximum surrogate is incremented by one, but one may also use different schemes to compute the surrogate, for example if one wants to reuse surrogates from deleted elements. According to the `SURROGATE` definition the single surrogate values are concatenated to build the compound surrogate *hsk*.

6.2 Insertion into the Fact Table: Lookup of Reference Surrogates

An insertion of a tuple into the fact table specifies the key dimension attributes (the dimension attributes of the leaf hierarchy level). For each dimension, however, the corresponding compound surrogate must be found.

This so called lookup is to be performed before the insertion of the tuple into the fact table because the tuple has to be extended by the corresponding compound surrogates for the dimensions (reference surrogates). For every dimension d_i, a direct access to the dimension table is performed to retrieve the corresponding *cs*.

Depending on the number of dimensions, a number of B*-Tree accesses is necessary and thus will decrease the insert performance. Especially for bulk loading, such a procedure may cause long insertion times, because the caching of the dimension tables may be not optimal. Thus, an alternative lookup concept has been implemented in Transbase® by loading a hash table with the dimension keys and corresponding compound surrogate for every dimension before inserting the fact table tuples. Then the lookup is very efficient, because only main memory accesses will be necessary in the hash tables.

6.3 Hierarchy Reorganization

Usually, dimension hierarchies are very stable. However, there are various application domains, where frequent hierarchy reorganizations, e.g., moving of sub-trees, occur and should be reflected in the data organization. Such operations require additional support by the DBMS as local operations on dimension tables now have immediate influence on the organization of the fact table. These operations are usually much more expensive. For example, reclassifying one product group to a different category will cause changes in the *hsks*. In order to have a correct physical clustering of the fact table all fact tuples corresponding to the changed product group have to be updated. Of course, arbitrary hierarchy reorganizations by data updates are supported, but it has to be kept in mind that the necessary fact table updates may be very time critical.

We have implemented similar methods as for star query processing, in order to support hierarchy reorganization and the corresponding fact table reclustering as good as possible. For example, we optimized multi-query box algorihtm methods that are necessary, if a number of hierarchy paths change and require corresponding reorganization of the fact table records. An alternative method is to delay the reorganization of the fact table until a background process uses idle time of the warehouse application for the reclustering. In this case, we introduce two h-surrogates, where one

contains the previous (hsk_1) and the other (hsk_2) contains the new value. hsk_1 is updated, when the corresponding fact table records are reorganized. If hsk_1 and hsk_2 both have the same value, no reclustering is necessary.

7 Query Processing with MHC & UB-Trees

In this section we discuss the basic star query processing for DW schemata with MHC and multidimensional index organization and its advantages. For the discussion in this paper, we restrict ourselves to standard "star queries".

7.1 Query Template

Figure 9 shows the SQL query template for simple ad hoc star-queries. The notation {X} represents a set of X objects. The template defines typical star queries and uses abstract terms that act as placeholders. Note that the queries that conform to this template generally have a structure that is a subset of the above template and they instantiate all abstract terms.

Our template is applied on a schema similar to the one in Figure 3, which is a typical star schema. It specifies the restrictions on the various dimensions, the star-join between the fact table, and the required dimension tables and the subsequent grouping and aggregation. In general any attribute (hierarchical, feature, or measure) can appear in a `GROUP BY` clause. However, most queries group the result by a number of hierarchical and/or feature attributes. Finally, there is an `ORDER BY` to control the order of the presented results.

```
SELECT      {D.h},{D.f},{FT.m},
            {AGGR(FT.m) AS AM}, {AGGR(D.h) AS AH},
            AGGR(D.f) AS AF}
FROM        FT,{D1},…,{Dn}
WHERE       FT.d1 = D1.h1  AND
            FT.d2 = D2.h1  AND      } Star-Join condition
            ...
            FTdn = Dn.h1   AND

            PRED(D1)   AND          } „local“ restrictions on the
            PRED(D2)   AND          } dimension tables
            ...                     }
            PRED(Dn)   AND          }
                                    } restrictions to measures
            PRED({FT.m})            }
GROUP BY    {D.h},{D.f},{FT.m}
HAVING      PRED({AM},{AH},{AF})
ORDER BY    <ordering fields>
```

Figure 9: Star Query Template

7.2 Star Query Processing with MHC

Processing of star queries as described above can roughly be divided into three major steps:

(1) Evaluation of dimension predicates

(2) Fetching result tuples from the fact table

(3) Residual joins, grouping and aggregation, sorting

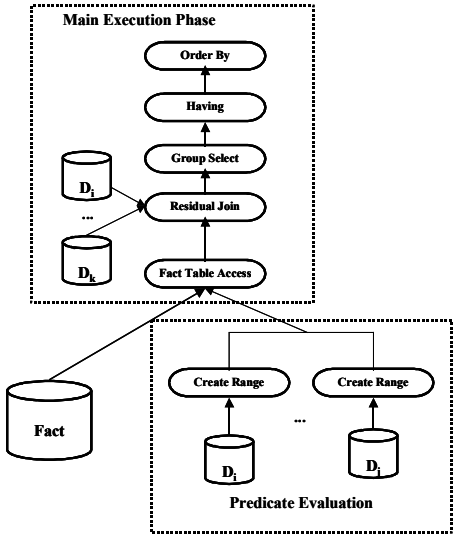


Figure 10: Standard Abstract Execution Plan

The first step evaluates the predicates on the dimension tables. The second and third step are often considered together as not all restrictions can be only evaluated on the fact table. In the following, we speak of two processing phases:

- Predicate Evaluation
- Main Execution Phase

In the presence of MHC and a multidimensional index for the organization of the fact table new optimizations can be applied to these phases. This is reflected already in the abstract execution plan (AEP) [Kar02] of Figure 10.

7.3 Predicate Evaluation and Fact Table Access

In the predicate evaluation phase the first benefit of HC can be observed. Instead of generating a large set of point restrictions, the local predicates on the dimensions usually result in a (small) set of range restrictions. This is especially true for predicates with hierarchical restrictions (cf. Figure 2 (b)) but also many feature restrictions will lead to ranges if there is some dependency between the defined hierarchy and the feature values. The details of the interval generation are discussed in [Kar02].

After the generation of the intervals per dimension, the combination of intervals is transformed into a number of query boxes that are executed by the *Fact Table Access*. At that point, the multidimensional organization of the fact table yields a major cost saving: relying on the clustering and the

efficient range query algorithm of the UB-Tree the number of I/O to fetch the required data is drastically reduced.

7.4 Main Execution Phase: Grouping and Aggregation

The Main Execution Phase joins the tuples that are fetched from the fact table with all necessary dimension tables (*Residual Join* operator). After the residual join the tuples are grouped (*Group-Select* operator), filtered again (*Having* operator) and sorted (*OrderBy* operator). This phase strongly benefits from the additional information which is encoded in the fact table tuples via the reference surrogates. The reference surrogates represent an encoding of the hierarchy path of each level member – therefore important optimizations can be realized for GROUPing on dimension fields. Especially, in many cases, a great deal of the grouping work can be done locally on the fact table tuples using prefixes of the reference surrogates.

The important point is that this pre-grouping step is done before the fact table tuples are joined with dimension tables. As the grouping operation typically greatly reduces tuple cardinality, the cost for the successive join operation is also greatly reduced. This technique is called *Hierarchical Pre-grouping*. In detail, if the GROUPing field belongs to hierarchy level h_k or is functionally dependent on it then the h-surrogate prefix $h_t/h_{t-1}/.../h_k$ is dynamically computed and the GROUP operation is done on that value.

A simple example illustrates the effects of the hierarchical pre-grouping method: Assume we have a DW with a time dimension (besides other dimensions) categorized by year – month – day and a well populated fact table. A query restricting the result to year 2001 (besides other restrictions) qualifies 100.000 fact table records. If the result has to be grouped w.r.t. month, we have to join 100.000 records with the time dimension table. When applying hierarchical pre-grouping, the number of join operations is reduced by a factor of 30, because all days of one month are aggregated to the month.

Our measurements with a five dimensional real world DW show an average reduction of the join cardinality by more than a factor of 100. This leads to an overall speedup of a factor of 4 to 7 for the grouping and residual join execution phase.

Figure 11 shows the final abstract execution plan used in Transbase®. Due to the effective pre-grouping steps, the costly overall residual join step is avoided if possible.

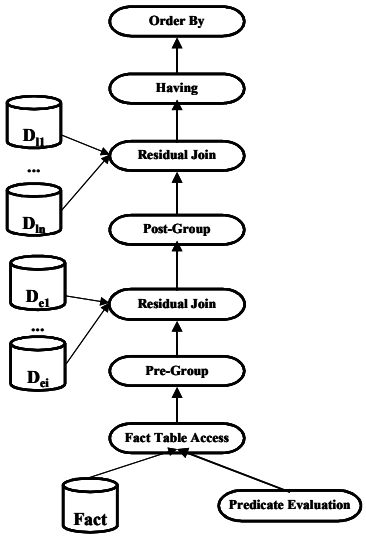


Figure 11: Abstract Execution Plan with Pre-Grouping

We do not enlarge on this optimization in this paper. For detailed information about hierarchical grouping please refer to [Pie03].

8 Evaluation of MHC

In this section we briefly want to illustrate the benefits of MHC and multidimensional indexing in the context of a real-world data warehouse application.

The measurements are performed on a two processor PC Pentium III, 750 MHz, with 256 MB RAM and 30 GB IDE hard disk. We used Windows 2000 as operating system and Transbase® with the MHC implementation as DBMS.

The DW schema of a large electronic retailer consists of a fact table with three dimensions *CUSTOMER*, *PRODUCT* and *DATE* and 3 measures: *quantity*, *value,* and *unit_price*. The *CUSTOMER* dimension contains 1,4 million records, *PRODUCT* consists of 27.000 products and the *DATE* dimension covers 7 years on day granularity. 15.543.380 records are stored in the snapshot of the fact table, amounting to 1,5 GB.

The query workload consisted of 220 ad hoc star queries from a real-world application. We classified the queries into three groups according to their selectivity on the fact table (i.e., number of tuples retrieved from the fact table):

- [0.0-0.1]: 0% to 0.1% of fact table, i.e., 0 to about 15K records
- [0.1-1.0]: 0.1% to 1% of fact table, i.e., 15K to 160K records
- [1.0-5.0]: 1.0% to 5.0% of fact table, i.e., 160K to 780K records

The goal of the performance evaluation was to measure three alternative query processing techniques:

- *STAR*: conventional star join processing without MHC and without multidimensional indexing; STAR uses secondary indexes that are created on the dimension keys of the fact table. The restrictions on the dimension tables are evaluated and the resulting dimension keys are used for index intersection on the

fact table. The resulting records are joined with the dimension tables, in order to perform grouping and get the final result. This is the typical processing of star queries in commercial DBMSs (e.g., *star transformation* in Oracle [Ora01]). This processing has two major steps: the index intersection and the tuple materialization. While the index intersection has largely been optimized (e.g., with bitmap indexes [NQ97]) the materialization of results is still the bottleneck of non-clustering indexes. Consequently, we neglect the index intersection time for STAR and just measure the time for fact record materialization, residual joins and grouping. Thus, the times for STAR have to be considered as lower bounds of the real processing time.

- *MHC*: applying MHC and multidimensional indexing to the fact table
- *OPT*: MHC with pre-grouping optimizations

For MHC and OPT the complete processing including index access is measured.

Table 1: Response time (in sec) for the three techniques for the three query classes

FT Sel. %	**[0.0-0.1]**			**[0.1-1.0]**			**[1.0-5.0]**		
	STAR	**MHC**	**OPT**	**STAR**	**MHC**	**OPT**	**STAR**	**MHC**	**OPT**
MIN	0	0	0	65	2	2	274	11	6
MAX	30	6	3	290	9	6	1219	47	27
MEDIAN	1	1	1	182	8	5	477	23	13
STD-DEV	4.9	1.2	0.5	75.6	3.1	1.6	346.0	14.1	7.9

Table 1 shows the response time analysis (in seconds) for the three alternative processing plans. As the three classes contain queries with different result set size and thus different response times we use the maximum, minimum, median time and the standard deviation to analyze the performance.

Our results show that the standard STAR processing is outperformed by our approaches. However, for small queries, i.e., the class [0.0-0.1], the speedup is below an order of magnitude. In general, for small result sets, the advantage of clustering over non-clustering is not that large. The picture changes drastically, when we consider larger queries (classes [0.1-1.0] and [1.0-5.0]), which are more typical for OLAP applications. The hierarchical clustering of MHC leads to an average speedup compared to STAR of 24 and with the additional optimization of pre-grouping an additional factor of about two is gained. Note also that STAR has a very high deviation in the response times for queries within one class. This is mainly for two reasons: (a) STAR performance deteriorates very fast as the fact table selectivity is increased and (b) since the fact table is not stored clustered the number of performed I/Os may differ significantly from one query to another. On the other hand, the deviation for MHC and OPT remains low, showing a much more stable behavior.

9 Summary

In this paper we presented the Transbase® DMBS supporting multidimensional hierarchical clustering (MHC). Multidimensional indexing and hierarchical clustering is combined for the primary organization of the fact table of a typical star schema. Transbase® is running on UNIX systems as HP-UX, SUN Solaris, AIX and Linux and on Windows platforms.

The integration is made almost transparent to the user: only when creating the dimension tables and the fact table the hierarchy and the clustering specification has to be provided. From then on, the system is automatically maintaining the hierarchical clustering and uses it for query processing.

We presented some of the implementation issues of MHC and advanced query optimization features enabled by MHC. The measurements on a real-world data warehouse illustrated that MHC can dramatically improve star query processing on such organized schemata.

The improvements of the query processing are not restricted to star schemata, also snowflake schemata are supported. Therefore, our approach is usable in many real data warehouse applications, even complex ones.

In the future we are implementing some advanced algorithms, in order to support complex expressions in aggregation functions, improve queries with non-clustering dimensions and support joins over multiple fact tables.

Bibliography

[ACN01] S. Agrawal, S. Chaudhuri, V. R. Narasayya: Materialized View and Index Selection Tool for Microsoft SQL Server 2000. SIGMOD Conference 2001

[Bay97] R. Bayer. The universal B-Tree for multi-dimensional Indexing: General Concepts. WWCA '97. Tsukuba, Japan, LNCS, Springer Verlag, March, 1997.

[CD97a] S. Chaudhuri, U. Dayal: An Overview of Data Warehousing and OLAP Technology. SIGMOD Record 26(1): 65-74 (1997)

[CD97b] S. Chaudhuri, U. Dayal: Data Warehousing and OLAP for Decision Support (Tutorial). SIGMOD Conference 1997: 507-508

[CI98] C. Y. Chan, Y. E. Ioannidis: Bitmap Index Design and Evaluation. SIGMOD Conference 1998: 355-366

[CS94] S. Chaudhuri, K. Shim: Including Group-By in Query Optimization. VLDB 1994: 354-366

[Des98] P. Deshpande, K. Ramasamy, A. Shukla, J. F. Naughton: Caching Multidimensional Queries Using Chunks. SIGMOD Conference 1998: 259-270

[Gra96] J. Gray, A. Bosworth, A. Layman, H. Pirahesh: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. ICDE 1996: 152-159

[GG97] V. Gaede and O. Günther. Multidimensional Access Methods. ACM Computing Surveys 30(2), 1997.

[GM95] A. Gupta, I. S. Mumick: Maintenance of Materialized Views: Problems, Techniques, and Applications. Data Engineering Bulletin 18(2): 3-18 (1995)

[Kim96] R. Kimball. The Data Warehouse Toolkit. John Wiley & Sons, New York. 1996.

[KR98] Y. Kotidis, N. Roussopoulos: An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees. SIGMOD Conference 1998: 249-258

[KS01] N. Karayannidis, and T. Sellis: SISYPHUS: A Chunk-Based Storage Manager for OLAP Cubes. DMDW 2001

[Kar02] N. Karayannidis, A. Tsois, T. Sellis, R. Pieringer, V. Markl, F. Ramsak, R. Fenk, K. Elhardt, R. Bayer: Processing Star Queries on Hierarchically-Clustered Fact Tables. VLDB 2002

[MRB99] V. Markl, F. Ramsak, R. Bayern: Improving OLAP Performance by Multidimensional Hierarchical Clustering. Proc. of the Intl. Database Engineering and Applications Symposium, pp. 165-177, 1999.

[NG95] P. E. O'Neil, G. Graefe: Multi-Table Joins Through Bitmapped Join Indices. SIGMOD Record 24(3): 8-11 (1995)

[NHS84] J. Nievergelt, H. Hinterberger, K. C. Sevcik: The Grid File: An Adaptable, Symmetric Multikey File Structure. TODS 9(1): 38-71 (1984)

[NQ97] P. E. O'Neil, D. Quass: Improved Query Performance with Variant Indexes. SIGMOD Conference 1997: 38-49

[OM84] J. A. Orenstein, T. H. Merret. *A Class of Data Structures for Associate Searching*. Proc. of ACM SIGMOD-PODS Conf., Portland, Oregon, 1984, pp. 294-305

[Ora01] Oracle 8i Documentation, 2001.

[Pie03] R. Pieringer, K. Elhardt, F. Ramsak, V. Markl, R. Fenk, R. Bayer, N. Karayannidis, A. Tsois, T. Sellis: Combining Hierarchy Encoding and Pre-Grouping: Intelligent Grouping in Star Queries. ICDE 2003

[Ram00] F. Ramsak, V. Markl, R. Fenk, M. Zirkel,K. Elhardt, R. Bayer.Integrating the UB-Tree intoa Database System Kernel. In VLDB2000,Proceedings of International Conference on Very Large Data Bases, 2000, Cairo, Egypt, 2000.

[Rou98] N. Roussopoulos: Materialized Views and Data Warehouses. SIGMOD Record 27(1): 21-26 (1998)

[Sam90] H. Samet. The Design and Analysis of Spatial Data Structures. Addison Wesley, 1990

[Sar97] S. Sarawagi: Indexing OLAP Data. Data Engineering Bulletin 20(1): 36-43 (1997)

[SDJL96] D. Srivastava, S. Dar, H. V. Jagadish, A. Y. Levy: Answering Queries with Aggregation Using Views. VLDB Conference 1996: 318-329

[Tra02] The TransBase HyperCube relational database system, available at: http://www.transaction.de/

[TT01] D. Theodoratos, A. Tsois: Heuristic Optimization of OLAP Queries in Multidimensionally Hierarchically Clustered Databases. DOLAP 2001.

[WB98] M. C. Wu, A. P. Buchmann: Encoded Bitmap Indexing for Data Warehouses. ICDE 1998: 220-230

[WOS01] K. Wu, E. J. Otoo, A. Shoshani: A Performance Comparison of bitmap indexes. CIKM 2001: 559-561

[Wu99] Ming-Chuan Wu: Query Optimization for Selections Using Bitmaps. SIGMOD Conference 1999: 227-238

[YL94] W. P. Yan, P-Å. Larson: Performing Group-By before Join. ICDE 1994: 89-100

[YL95] W. P. Yan, P.-Å. Larson: Eager Aggregation and Lazy Aggregation. VLDB Conference 1995

[Zah00] Markos Zaharioudakis, Roberta Cochrane, George Lapis, Hamid Pirahesh, Monica Urata: Answering Complex SQL Queries Using Automatic Summary Tables. SIGMOD Conference 2000:

[ZSL98] C. Zou, B. Salzberg, and R. Ladin: Back to the Future: Dynamic Hierarchical Clustering. Proc. Of ICDE, 1998, pp. 578-587.

Partitioned B-trees – a user's guide

Goetz Graefe
Microsoft Corporation
Redmond, WA 98052-6399

Abstract: A recent article introduced partitioned B-trees, in which partitions are defined not in the catalogs but by distinct values in an artificial leading key column. As there usually is only a single value in this column, there usually is only a single partition, and queries and updates perform just like in traditional B-tree indexes. By temporarily permitting multiple values, at the expense of reduced query performance, interesting database usage scenarios become possible, in particular for bulk insert (database load). The present paper guides database administrators to exploiting partitioned B-trees even if they are not implemented by their DBMS vendor.

1 Introduction

The essence of partitioned B-tree indexes [G 03] is to maintain partitions within a single B-tree, by means of an artificial leading key column, and to reorganize and optimize such a B-tree online using, effectively, the merge step well known from external merge sort. This key column should be an integer of 2 or 4 bytes. By default, the same single value appears in all records in a B-tree, and the techniques proposed here rely on temporarily permitting and exploiting multiple alternative values. If a table or view in a relational database has multiple indexes, each index has its own artificial leading key column. The values in these columns are not coordinated or propagated among the indexes. In other words, each artificial leading key column is internal to a single B-tree, such that each B-tree can be reorganized and optimized independently of all others.

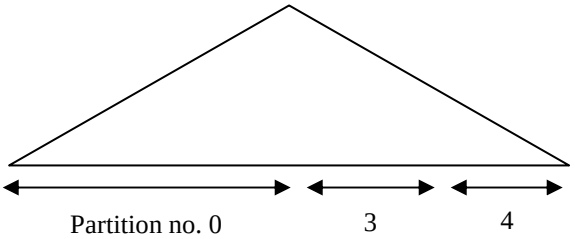


Figure 1. B-tree with partitions

The leading artificial key column effectively defines partitions within a single B-tree. Each existing distinct value implicitly defines a partition, and partitions appear and vanish automatically as the first or last record with a specific value are inserted and deleted. Partitioned B-trees differ from traditional horizontal partitioning using a separate B-tree for each partition in an important way. Most advantages of partitioned B-trees depend on partitions (or distinct values in the leading artificial key column) being created and removed very dynamically. In a traditional implementation of partitioning, each creation or removal of a partition is a change of the table's schema and catalog entries, which re-

quires locks on the table's schema or catalog entries and thus excludes concurrent or long-running user accesses to the table, as well as forcing recompilation of cached query and update plans. If partitions are created and removed as easily as inserting and deleting rows, smooth continuous operation is relatively easy to achieve.

Queries must probe each partition; while multiple partitions exist, query performance is reduced. Suitable algorithms for searching indexes with multiple partitions have been described elsewhere [LJB 95] and are not considered here in detail. Database management systems vary in their ability to generate and execute such plans; experimentation is required to ensure acceptable query plans are chosen when probing multiple partitions. For sorted scans, the ideal plan merges multiple ordered scans, one per partition. If indexes usually imply statistics and histograms, selectivity estimation may be improved by separate statistics on the user's column only, i.e., without the artificial leading key column.

While partitioned B-trees are very useful for sorting, index creation, and bulk insertion [G 03], the present paper focuses on bulk insertion. Adding a large amount of data to a large, fully indexed data warehouse so far has created a dilemma between dropping and rebuilding all indexes or updating all indexes one record at a time, implying random insertions, poor performance, a large log volume, and a large incremental backup. Partitioned B-trees resolve this dilemma in most cases. Multiple indexes may exist and are maintained correctly, without the need for expensive random insertions into each B-tree. Note that partitioned B-trees offer these advantages without special new data structures, which means that B-trees as implemented and available today can be adapted and used as partitioned B-trees. The present paper provides guidance on how to do so.

2 Comparison of bulk insertion strategies

As an example, assume a table with a clustered index and no other indexes. Assume also that this table already contains 100M rows on 1M pages, and that another 1M rows must be added. This 1% could represent one eight-hour shift within a month, one day within a quarter, or one week within a two-year period – thus, this is a typical scenario in a data warehouse. Finally, assume that the clustered index is not sorted on a time attribute, i.e., integrating the new data into the existing clustered index will require a lot of key searches and random I/Os. If the operation modifies (reads and writes) 1M pages, a database server performing 1,000 I/Os per second will require 2,000 seconds or about ½ hour of dedicated server time. Note that this strategy will require either a large number of key locks or it will lock the entire table for the entire time, completely preventing concurrent updates and queries.

If the database server employs one of the optimized strategies that pre-sort the change set before applying it to an index, we may presume that I/O operations can often move multiple pages at once and will thus be 2-4 times faster than random I/O. Thus, the time to apply the bulk insert is reduced to about ¼ hour. Again, this strategy is likely to lock the entire table for practically the entire time.

In fact, it might be faster to drop the pre-existing index, add the new data, and then build an entirely new index. Adding 1M rows or about 10K pages to a heap is very fast, requiring about 10 seconds. Presuming index creation employs an external merge sort with a single merge step, index creation will require I/O for about 4M pages. Presuming that

sorting uses large I/Os with bandwidth 4 times higher than random I/Os, this strategy will require also about ¼ hour. Even if the update plan and index construction do not lock the entire table and clustered index, concurrent queries and updates will perform extremely poorly due to the missing index.

Now presume that the clustered index has an artificial leading key column, and that the value in this column for all pre-existing rows is 0. The fastest way to insert 1M rows is to insert all of them with a new value for this column, say 1. If so, the pre-existing rows and the new rows will be in separate partitions within the clustered index. All new rows can be appended to the pre-existing B-trees, which permits not only packing new records very densely but also permits optimizations for the insertion logic (no need for a root-to-leaf search for each record), for write I/O (large writes), and for logging (log entire pages rather than single records). If 1M new rows require 10K new pages, this insertion can complete in 10 seconds or less.

The insert set ought to be sorted because appending to a B-tree is faster than inserting into a B-tree. For truly large inserts, the sort operation might require external sorting and thus I/O of its own. In this case, it might make sense to append multiple smaller partitions, each one requiring only an in-memory sort. The reorganization that merges the newly appended partitions into the pre-existing main partition is equally efficient for any small number of new partitions.

Once all records have been inserted into the database, the new partitions ought to be merged into the pre-existing main partition. The important observations are that (1) the pre-existing keys and pages do not need to be locked and are available for concurrent queries and updates at all times, (2) the index is never dropped and concurrent operations proceed with normal performance, (3) the new data are available for queries and updates immediately after the insertion is complete, and (4) partitions can be merged in many small transactions, each transaction merging only a small range of keys from the new partition into the main partition. Because these transactions are small and fast, they are unlikely to interfere very much with concurrent transaction processing.

3 Example SQL commands

The following code shows some SQL commands in order to illustrate the suggestions above. The SQL code below includes a query as it should be used in all application queries – better yet, the table should be queried through a view that adds this predicate to all queries and it should be updated to a view that enforces the default value for the artificial leading key column in all insertions.

```
Create table tbl (a varchar(20), b float, c datetime, …) -- initial, empty table
Alter table tbl add column x int not null check (value >= 0) default 0
Insert into tbl (x, a, b, c) select 0, … from … -- initial 100M rows, partition 0
Create clustered index clu on tbl (x, a) -- index with artificial leading key column
Insert into tbl (x, a, b, c) select 1, … from … -- add 1M rows, partition 1
Insert into tbl (x, a, b, c) select 2, … from … -- add another 1M rows
…
Insert into tbl (x, a, b, c) select 5, … from … -- add the 5th 1M rows
Select … from tbl where x in (0, 1, 2, 3, 4, 5) and … -- application query or view
```

```
While (exists (select * from tbl where x in (1, 2, 3, 4, 5)) -- reorganization
Begin -- reorganize about 100 rows at a time
    If (exists (select * from tbl where x = 1))
        Update tbl set x = 0 where x in (1, 2, 3, 4, 5) and a <= (select max (a) from
            (select top 20 a from tbl where x = 1 order by a) as t)
    …
    If (exists (select * from tbl where x = 5))
        Update tbl set x = 0 where x in (1, 2, 3, 4, 5) and a <= (select max (a) from
            (select top 20 a from tbl where x = 5 order by a) as t)
End
```

The set of existing partitions (or values in the artificial leading key column) could be administered using a small auxiliary table and a referential constraint, which permits leaving the view definition in place as partitions appear and vanish.

If there is more than one index, e.g., non-clustered indexes, there ought to be a separate artificial key column for each of them. When inserting data, appropriate new values are assigned to all of them, ensuring fast append logic for all B-tree indexes.

4 Summary and conclusions

In summary, an artificial leading key column in each B-tree index enables partitioned B-trees, which permit not only fast bulk insertion but also fast memory-adaptive external merge sort as well as improved index creation [G 03]. If not implemented by the database management system vendor, artificial leading key columns can be declared explicitly by database administrators adapting the sample code provided above, which can also be adapted for bulk deletion from tables with one or multiple indexes.

The important value for bulk insertion is that the actual insertion step becomes very fast, and that pre-existing indexes are available for queries immediately after the insertion command completes. While multiple partitions exist, query performance is reduced, and online reorganization might require more log space than traditional import methods. These disadvantages must be judged against the fast insertion times; in many practical cases, partitioned B-trees based on artificial leading key columns and online reorganization can be more attractive than any of the alternatives. If this brief paper helps database administrators save valuable time developing import strategies using partitioned B-trees, it has achieved its purpose.

References

[G 03] Goetz Graefe: Sorting and Indexing with Partitioned B-trees. Conf. Innovative Data Systems Research, Asilomar, CA, Jan. 2003.

[LJB 95] Harry Leslie, Rohit Jain, Dave Birdsall, Hedieh Yaghmai: Efficient Search of Multi-Dimensional B-Trees. VLDB Conf. 1995: 710-719.