# Workflowevolution

Michael Schwipps

14. Februar 2006

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Einleitung

Diese Arbeit beschäftigt sich im Rahmen des Seminars „Schema Evolution“ mit dem Thema der Workflowevolution. Das Seminar wird von der Abteilung Datenbanken am Institut für Informatik an der Universität Leipzig organisiert.

Die Workflow Management Coalition [19] definiert einen Workflow wie folgt: „The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.“ [13] Der Arbeitsablauf(engl.: Workflow) konzentriert sich auf den reinen Ablauf der Aktivitäten. Die Ausführung der Aktivitäten(engl.: Tasks) bleibt außen vor, d.h. das die Aktivitäten durch den Workflow veranlasst werden, aber deren Abarbeitung selbst nicht durch den Workflow beschrieben wird. Das ursprüngliche Interesse an Workflows und ihrer computerunterstützten Ausführung kam vor allem aus Unternehmen. Forschung und Entwicklung findet deshalb auch in den Wirtschaftswissenschaften statt. Zur Stärkung ihrer Wettbewerbsposition fordern Unternehmen, dass die Arbeitsabläufe effizient sind. Effiziente Arbeitsabläufe werden allerdings auch in anderen Bereichen(z.B. Behörden) erwartet. Die Definition der Effizienz, insbesondere deren wirtschaftliche Konsequenzen und alle weiteren betriebswirtschaftlichen Zusammenhänge werden hier nicht betrachtet. Die Begriffe Arbeitsablauf, Prozess und Workflow werden im Folgenden synonym verwendet. In Abbildung 2 ist ein Beispiel für einen Workflow zu sehen. Dieser spezifiziert den Bau eines PCs.
Ähnlich wie bei Datenbanken wird zwischen dem Schema und den Instanzen unterschieden. Das Schema definiert die statischen Aspekte eines Workflows. Es wird in der Literatur auch als Workflowtyp bezeichnet und beschreibt den Workflow. Das Schema enthält die Metadaten. Die Instanzen bilden die ausgeführten Prozesse. Instanzen werden auch als Fälle bezeichnet. Sie laufen nach einem Schema ab. Workflows bzw. Schemata können hierarchisch aufgebaut sein, d.h. eine Aktivität eines übergeordneten Workflows startet einen Subprozess.

Für die automatisierte Steuerung der Arbeitsabläufe werden so genannte Workflow-Management-Systeme(WfMS) eingesetzt. Weitere Einsatzgebiete der WfMS sind die computerunterstützte Analyse, Planung, Dokumentation und Auswertung von Prozessen. WfMS unterstützen strukturierte Prozesse. Im Gegensatz dazu geben Groupware-Systeme die Möglichkeit unstrukturierte Prozesse zu bearbeiten. Die auszuführenden Aktivitäten sind in der Regel nicht Teil des WfMS. Die Systeme entkoppeln den Datenfluss der Applikation und den Kontrollfluss der Prozesse. Sie ermöglichen auch das Einbinden von nicht rechnergestützten Aktivitäten und stellen ein Verbindungsglied zwischen den den einzelnen Aktivitäten zugeordneten Applikationen dar. Auf die Implementierung von WfMS bzw. die technische Umsetzung der Evolution wird in dieser Arbeit nicht bzw. nur am Rande eingegangen.
Folgenden Erwartungen werden (laut [11]) mit dem Einsatz von WfMS verknüpft.

- *höhere Qualität der Verarbeitung* Die höhere Qualität der Verarbeitung entsteht durch das maschinelle Starten von Aktivitäten und die von persönlichen Präferenzen unabhängigen Entscheidungen. Es können weder Aufgaben vergessen werden, noch werden unnötige Aktivitäten durch das WfMS gestartet.

- *schnelleres Bereitstellen benötigter Informationen* Im Gegensatz zur physischen Verteilung von Dokumenten(z.B. Briefen) ist die Übermittlung von bearbeiteten Dokumenten

mittels WfMS deutlich schneller möglich, da sie sofort dem nächsten Mitarbeiter zu Verfügung steht, wenn der vorherigen Mitarbeiter seine Aktivität beendet hat.

- *schnellere Abwicklung von Vorgängen, höherer Durchsatz* Diese Erwartungen sind eine Konsequenz aus der vorhergenannten schnelleren Bereitstellung von Informationen.
- *besserer Kunden-Service* WfMS bieten beispielsweise die Möglichkeit dem Kunden den aktuellen Bearbeitungsstatus eines Auftrags mitzuteilen. Bei Paketzustellern ist dies inzwischen üblich.
- *erhöhte Produktivität, reduzierte Ausführungskosten* Die erhöhte Produktivität ergibt sich aus den geringe Wartezeiten für das Bereitstellen von Informationen. Die eingesetzten Ressourcen und Mitarbeiter werden besser ausgelastet.
- *bessere Überprüfbarkeit von Abläufen* Da sämtliche Workflows durch das WfMS gestartet und überwacht werden, kann auch protokolliert werden, wann die einzelnen Aufgaben erledigt wurden.
- *bessere Integration der Infrastruktur / vorhandener Datenbanken* Im Gegensatz zu nicht computerunterstützten Arbeitsabläufen, ist es dem WfMS möglich, auf vorhanden Datenbanken elektronisch zuzugreifen.
- *besseres Verständnis des Produktionsprozesses* Dieser Punkt ergibt sich zwangsweise aus der Notwendigkeit einen Arbeitsablauf im WfMS zu definieren. Schlecht motivierten Änderungen des Arbeitsablauf steht der Aufwand entgegen diese dem WfMS mitzuteilen.
- *Flexibilität hinsichtlich Umstellung / Anpassung der Abläufe an geänderte Anforderungen* Die Definition von Workflows in WfMS ist arbeitsaufwendig und verursacht dementsprechend Kosten im Unternehmen. Von dem WfMS wird eine möglichst einfache Anpassung der Prozesse erwartet.

Dem stehen folgende Befürchtungen und Probleme gegenüber

- *Kontrolle und Überwachung der Mitarbeiter* Sie kann zu einer schlecht motivierten Belegschaft führen. Mögliche Konsequenzen sind zunächst das Umgehen des WfMS und anschließend ein Verlust an Produktivität.
- *Funktionsdefizite* Sie können beispielsweise dann auftreten, wenn Schnittstellen zu externen Systemen fehlen oder nicht alle denkbaren bzw. umzusetzenden Kontrollflüsse durch das WfMS unterstützt werden.
- *unzureichende Flexibilität* Sie ist dann gegeben, wenn die Erwartungen an die Flexibilität nicht erfüllt werden können.
- *Umstellungs- und Integrationsprobleme* Sie entstehen beispielsweise durch fehlende Schnittstellen auf bestehende Systeme. Diese sind unabhängig davon, ob die Schnittstellen auf Seiten des WfMS oder des externen Systems nicht existieren. Fehlende Schnittstellen führen dazu, dass die Systeme nicht miteinander interagieren können. Umstellungsprobleme treten nur bei Einführung des WfMS auf. Integrationsprobleme bestehen dauerhaft.

In einer sich schnell ändernden Umwelt ist die Anpassung und Änderung von Geschäftsprozessen für Unternehmen zunehmend von Bedeutung. Lassen sich die Abläufe in den Verwaltungssystemen nicht schnell genug an die Erfordernisse anpassen, zwingen sie die Anwender dazu, die Systeme zu umgehen bzw. falsch zu nutzen. Derartige Aktionen untergraben die Zuverlässigkeit des Systems und machen beispielsweise die Überwachung von Aufgaben und deren statistische Auswertung unnötig schwierig wenn nicht gar unmöglich. Es ist also erforderlich zunächst zu untersuchen, in welchem Rahmen Anpassungen notwendig sind, inwieweit sie vermieden werden können und welche Stolpersteine zu beachten sind.
Die Änderung eines Workflows genauer dessen Schemas wird als Evolution bezeichnet. Gibt es Instanzen des Schemas, müssen diese in Abhängigkeit von der Änderungsstrategie ebenfalls angepasst werden. Die ökonomischen Zusammenhänge und Konsequenzen der Evolution werden im Rahmen dieser Arbeit nicht betrachtet.
Exceptions bzw. Ausnahmen sind dazu gedacht unerwartete und unerwünschte Situationen abzufangen bzw. zu behandeln. Sie trennen die Prozess- von der Ausnahmebehandlungslogik, um so das Prozessdesign übersichtlicher und lesbarer zu gestalten. Exceptions können beispielsweise aus technischen Störungen, wie Strom- oder Netzwerkausfällen, oder menschlichen Fehler resultieren. Auf Ausnahmen muss in der Regel sofort und im Einzelfall reagiert werden. Die Behandlung erfolgt entweder durch Weitergabe an einen übergeordneten Workflow oder durch spezielle Aktionen wie z.B. einem Neustart oder einem Rollback der Aktivität.
Eine weitere Form der Änderung von Workflows bilden die so genannten ad-hoc Workflows. Die leider sprachlich schlecht von ad-hoc (momentane) Änderungen abgegrenzt sind. Ad-hoc Workflows geben dem Endanwender die Möglichkeit im Bedarfsfall ein Workflowschema zu definieren und zu instantiieren. In der Praxis war das Erfordernis dieser Möglichkeit beim Aufkommen von WfMS eine sehr verbreitete Meinung. Damals wurde zwischen strukturierten, semi-strukturierten und unstrukturierten Prozessen unterschieden. Beispielsweise lassen sich kreative Prozesse nur schwer bis ins letzte Detail (vor-)strukturieren und damit durch ein WfMS ausführen. Mit so genannten ad-hoc Workflows sollte der Anwender in die Lage versetzt werden, diesem Umstand Rechnung zu tragen. Jedoch hat sich dieses Vorgehen nicht durchgesetzt. Im späteren Verlauf folgten dann die bereits erwähnten Groupware Systeme, die diese Art von Prozessen unterstützen. Wenn hier im Folgenden von ad-hoc Änderungen gesprochen wird, ist die momentane Änderung eines Workflowschemas gemeint. Eine Ad-hoc Änderung betrifft ein konkretes Workflowschema, für das eine Instanz erzeugt wird. Genau für diese Instanz erfolgt dann eine Änderung. Für andere gleichzeitig laufende Instanzen oder zukünftige Instanzen spielt die Änderung keine Rolle. Im Unterschied zu Exceptions werden hier erwartete Einzelfälle (z.B. Sonderanfertigungen eines Produkts) eingeordnet.

Diese Arbeit setzt sich zum Ziel auf die Grundlagen zweier theoretischer Workflowmodelle einzugehen, die Möglichkeiten der Workflowevolution zu klassifizieren, deren Rahmenbedingung zu spezifizieren und Ansätze der Evolution in Standard und WfMS-Implementierungen zu beleuchten. Nach dem in Abschnitt 2 ein Vergleich verschiedener Definitionsansätze erfolgt, geht es in Kapitel 3 mit der Beschreibung der Änderungsmöglichkeiten weiter.

# 2 Workflowmodelle

In diesem Abschnitt sollen einzelne Ansätze zur Workflowmodellierung verglichen werden. Aufgrund dieser unterschiedlichen Ansätze unterscheiden sich dann auch die Änderungs- bzw.

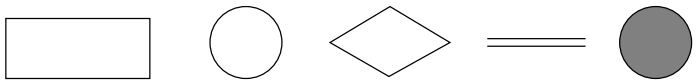

Abbildung 1: Flusssymbole: Aktivität, totale Verzweigung, bedingte Verzweigung, Start-/Stopsymbol, iterative Vereinigung

Evolutionsmöglichkeiten.
Allen Ansätzen ist gemein, dass die auszuführenden Aktivitäten als atomar angesehen werden.

## 2.1 Flussdiagrammbasiertes Modell

Der im folgenden betrachtete Ansatz (nach Casati [5]) orientiert sich an Flussdiagrammen. Es liegt sowohl eine formale Definition als auch eine graphische Notation für die einzelnen Elemente vor.
Im folgenden soll kurz die graphische Notation der Elemente des Diagramms (siehe Abbildung 1) beschrieben werden.

- *Aktivitäten* werden als Rechteck dargestellt.
- *Konnektoren* sind nur für Verzweigung(fork) oder Vereinigung(join) definiert.
- *Start- und Endeknoten* haben ein spezielles Symbol

Verzweigungen haben genau einen Vorgänger und mehr als einen Nachfolger. Es gibt bedingte (conditional) und unbedingte (total) Verzweigungen. Bei der bedingten Verzweigung werden nur diejenigen Nachfolger aktiviert, deren Bedingung erfüllt ist. Sie werden als Raute dargestellt. Bei der unbedingten Verzweigung werden alle Nachfolger aktiviert. Ihre Darstellung erfolgt als Kreis. Verzweigungen können den Beginn einer parallelen Abarbeitung von Aktivitäten markieren.
Vereinigungen haben einen Nachfolger und mehr als einen Vorgänger. Definiert sind iterative und totale(gesamte/gemeinsame) Vereinigungen. Bei einer iterativen Vereinigung wird der Nachfolger bei jeder Beendigung eines Vorgängers aktiviert. Sie wird graphisch als gefüllter Kreis dargestellt. Bei einer totalen Vereinigung wird gewartet, bis alle Vorgänger beendet sind. Anschließend wir der Nachfolger aktiviert. Ihre Darstellung erfolgt als leerer Kreis. Totale Vereinigungen synchronisieren Parallelverarbeitung.

Als anschaulicher Beispiel diene hier Abbildung 2. Zyklen bzw. Schleifen und damit das mehrfache Ausführen von Aktivitäten sind möglich. Neben der graphischen Notation gibt es, wie bereits angedeutet, eine formale Definition eines Workflows. Ein Workflow $W$, wird als Quadrupel $\Sigma^W =< Tasks^W, Vars^W, \sigma^W, \varsigma^W >$ beschrieben. $Task^W$ ist die Menge der Aktivitäten von $W$ und $Vars^W$ ist die Menge der Variablen von $W$. Die Nachfolgerfunktion $\sigma$ verbindet jede Aktivität $t$ aus $Tasks^W$ mit einer Menge von Aktivitäten aus $W$. Die Bedingungsfunktion $\varsigma$ ordnet jedem Task aus $W$ eine Bedingung zu. Die beiden Funktionen $\sigma$ und $\varsigma$ definieren die Flussstruktur (engl.: Routing). Verkürzt dargestellt bedeutet dies, dass nach Beendigung einer Aktivität $t$ mittels $\sigma$ alle Nachfolgeaktivitäten ermittelt werden. Für jede Nachfolgeaktivität $p$ wird nun mittels der Bedingungsfunktion $\varsigma(p)$ geprüft, ob $p$ ausgeführt werden soll.
Die Funktionen $\sigma$ und $\varsigma$ müssen einige Bedingungen erfüllen, damit der Workflow syntaktisch korrekt ist. Workflows gelten unter folgenden Voraussetzung als korrekt:

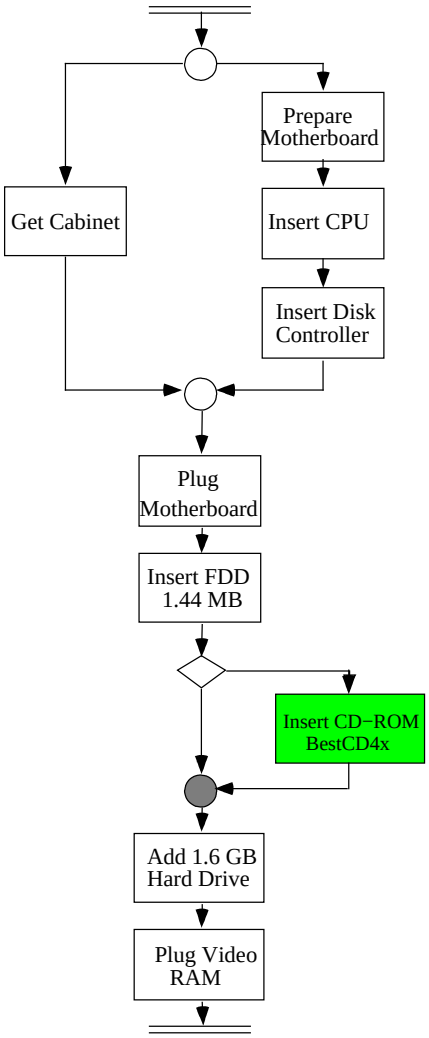


Abbildung 2: erstes PC-Bau-WF-Schema

- *transitive Hülle* der Nachfolgerfunktion: Die transitive Hülle der Nachfolgerfunktion $\sigma^*$ wird wie folgt definiert: $\sigma^*(t) = \{p : (p \in \sigma(t)) \vee (\exists s : s \in \sigma(t) \wedge p \in \sigma(s))\}$
- *Erreichbarkeit* Sie fordert das jede Aktivität $t$ ausgehend vom Startknoten erreichbar ist. $t \in \sigma^*('start')$
- *Vorgängerfunktion* Die Vorgängerfunktion definiert für einen Task $s$ die Menge der Tasks $\pi(s)$ die $s$ als Nachfolger haben. $\pi(s) = \{t : s \in \sigma(t)\}$

Ein Workflowschema ist korrekt, genau dann wenn alle Aktivitäten der Flussstruktur erreichbar sind. Als Konsequenz aus der Erreichbarkeit ergibt sich, dass ein Pfad existiert, der den Start- mit dem Endeknoten verbindet.

## 2.2 Petrinetzbasiertes Modell

In [1, 2, 3] bauen Workflows auf Petrinetzen auf. Dazu wird ein eigenes WF-Netz definiert. WF-Netze sind annotierte, stark verbunden Stellen-/Transitionsnetze. Sie sind korrekt, genau

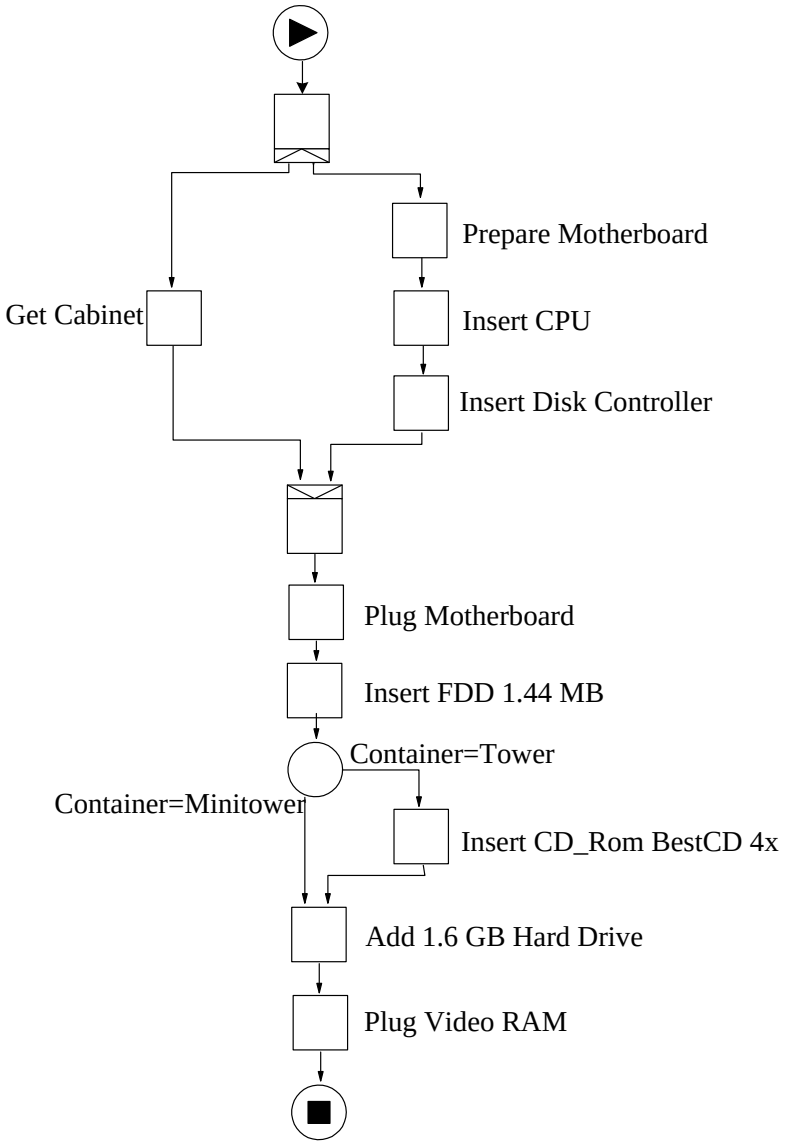


Abbildung 3: PC-Bau mit YAWL-symbolen (siehe Abbildung 9)

dann wenn sie sicher sind, korrekt beendet werden und keine toten Transitionen haben. Für eine vollständigere Entwicklung von WF-Netzen aus Petrinetzen sei auf [1] verwiesen.

Verschiedene Aspekte eines Geschäftsprozesses werden als Perspektiven bzw. Sichten bezeichnet. Es werden folgenden Sichten unterschieden.

1. *Prozesssicht* In dieser Perspektive erfolgt die Definition des Workflowschemas.

2. *Organisationssicht* In dieser Sicht werden die Organisationsstruktur und die Beteiligten festgelegt. Es werden die Beziehung zwischen Rollen, Gruppen und anderen organisatorischen Aspekten (z.B. Verantwortlichkeiten und Verfügbarkeiten) beschrieben. Rollen sind aufgrund funktionaler Aspekte klassifizierte Ressourcen. Gruppen sind Ressourcen, die auf Basis organisatorischer Aspekte klassifiziert werden. Die Ressourcen reichen dabei von beteiligten Menschen (z.B. Mitarbeitern) bis zu eingesetzten Geräten. Auf sie

wird nur über die definierten Gruppen und Rollen zugegriffen.

3. *Informationssicht* In der Informationsperspektive dreht sich alles um die Steuer(engl.: control)- und Produktionsdaten. Steuerdaten werden zu Workflowmanagementzwecken benötigt, z.B. für Entscheidungen bei Verzweigungen. Produktionsdaten (z.B. Dokumente oder Formulare) werden in den Aktivitäten benötigt und haben keinen Einfluss auf das Routing.

4. *Operationssicht* Diese Sicht beschreibt die elementaren Operationen, die durch Ressourcen und Applikationen ausgeführt werden. In der Prozesssicht werden diese Operationen typischerweise zum Erstellen, Lesen und Ändern der Steuer- und Produktionsdaten in der Informationssicht genutzt. Die meisten Operationen werden durch Anwendungsprogramme implementiert.

5. *Integrationssicht* Die Integrationssicht verbindet die anderen vier Sichten. Die Aktivitäten und Workflowprozessdefinitionen der Prozesssicht werden mit den Rollen, Gruppen und Ressourcen der Organisationssicht verbunden. Außerdem werden die Daten der Informationssicht und die Operationen der Operationssicht integriert.

Durch den Ansatz der Perspektiven ändert sich auch der Katalog der durch Änderungen am Workflow auftretenden Fehler. Die grundsätzliche Unterscheidung zwischen semantischen und syntaktischen Fehlern bleibt allerdings bestehen. Hinzu kommt beispielsweise die Differenzierung von transienten und permanenten sowie single- und multi-perspektiven Fehlern. Transiente Fehler sind Fehler, die ausschließlich in transformierten Instanzen eines Workflows auftreten und deren Ursache in einer unsachgemäßem Transformation begründet liegt. Permanente Fehler sind demgegenüber Fehler, die unabhängig von der Transformation auftreten. Singleperspektive Fehler treten nur in einer Perspektive auf. Multiperspektive Fehler betreffen mehrer Perspektive.

# 3 Änderungsstrategien und Evolutionsansätze

Wie bereits in der Einleitung (Abschnitt 1) erwähnt, gibt es unterschiedliche Ursachen die eine Anpassung von Arbeitsabläufen erforderlich machen. In diesem Abschnitt sollen zunächst die Änderungsstrategien klassifiziert werden. Anschließend wird dann auf zwei Ansätze zur Evolution eingegangen.

## 3.1 Klassifizierung

Änderungsstrategien lassen sich, wie in Abbildung 4 zu sehen, in unterschiedliche Klasse einteilen. Die einfachen Lösungen lassen sich wie folgt weiter aufteilen.

- *Abort* bezeichnet ein Vorgehen, bei dem alle laufenden Workflowinstanzen abgebrochen werden. Die bisher erfolgten Arbeitsschritte und erstellten Produkte werden dadurch wertlos. Das auch als „fordward recovery“ bezeichnete Verfahren ist das einzige, bei dem das WfMS nicht mehrere Versionen kennen muss. Es ist also aus technischer Sicht keine Versionierung erforderlich.

- *Flush* fordert, das zunächst alle Workflowinstanzen, die nach dem bestehenden Schema angefangen wurden, auch nach diesem Schema beendet werden müssen. Während dessen

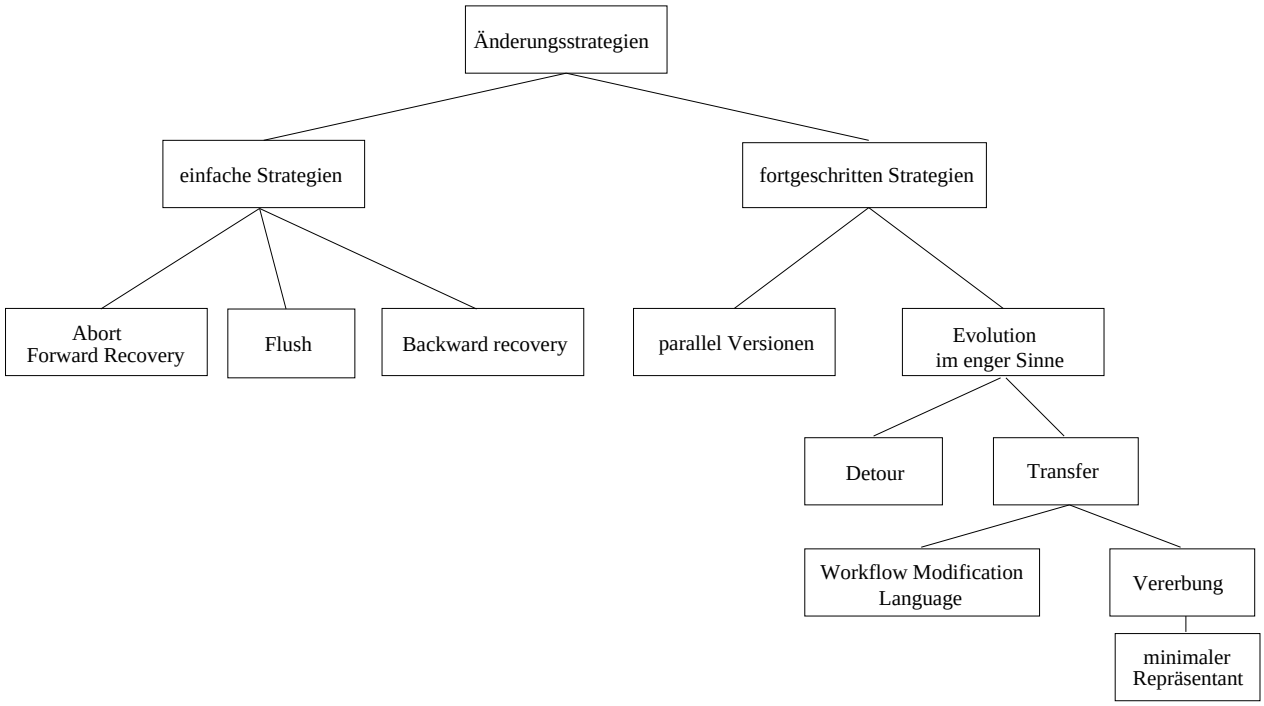


Abbildung 4: Änderungsstrategien

dürfen keine neuen Instanzen gestartet werden. Erst wenn alle Instanzen abgearbeitet sind, werden neue Instanzen nach neuem Schema gebildet.

- *Backward recovery* bezeichnet ein Verfahren, bei dem zunächst die bereits erfolgten Arbeitsschritte rückabgewickelt werden. Anschließend werden die Instanzen nach dem neuen Schema erneut ausgeführt.

Die genannten Varianten sind aber oft nicht möglich. So können beispielsweise gesetzliche Änderungen eine (im Verhältnis zur Workflowlaufzeit) sofortige Änderungen erfordern. Der Abbruch eines Prozesses ist in der Regel mit dem Verlust von Arbeitsergebnissen verbunden. Eine Rückabwicklung (engl.: rollback) ist gerade bei physischen Aktivitäten häufig nicht möglich bzw. unter ökonomischen Gesichtspunkten nicht sinnvoll.

Als fortgeschrittene Strategie lassen sich folgende Verfahren kategorisieren.

- *Parallele Versionen* treten dann auf, wenn es zulässig ist neue Workflowinstanzen nach neuem Schema zu starten, während bestehende Instanzen noch nach dem alten Schema arbeiten.

- *Transfer* bezeichnet die Möglichkeit bestehende Instanzen (eines alten Schemas) in ein neues Schema zu überführen.

- *Detour* ist für eine Sonderbehandlung von Fällen gedacht, die zwar nach einem neuen Schema abgearbeitet werden sollten, aber nicht überführt werden können. Das Verfahren führt neben dem alten und dem neuen Schema noch mindestens ein weiteres Schema ein, nach dem dann eine Menge von Instanzen abgearbeitet wird.

Die Klassifizierung ist nicht eindeutig, so spricht Dadam in [6] nur von Workflowevolution, wenn laufende Prozesse geändert werden. Der in Abbildung 4 aufgeführte Punkt „minimaler Repräsentant“ beschreibt ein Verfahren, wie der Transfer auf Vererbungsbasis effizienter definiert werden kann.

In [5] wird zwischen statischer und dynamischer Workflowevolution unterschieden.
Die statische Workflowevolution bezieht sich auf die Workflowdefinition, das heißt auf Änderungen am Schema. Das Entwurfswerkzeug muss Möglichkeiten zur Verfeinerung von Workflows bieten. Darüber hinaus sollte es sicherstellen, dass die Entwürfe syntaktisch korrekt sind.
Die dynamische Workflowevolution bezieht sich auf die Änderung der Schemata laufender Workflowinstanzen. Da nur Detour und Transfer Prozessinstanzen ändern, sind die dynamische Workflowevolution aus [5] und der Workflowevolutionsbegriff aus [6] identisch.

Aalst unterscheidet darüber hinaus zwischen momentanen (zeitweisen/ad-hoc) Änderungen und dauerhaften. Bei momentanen Änderungen soll nur eine Instanz des Prozesses geändert werden. Bei dauerhaften Änderungen werden mindestens alle neue Instanzen eine Workflows nach dem neuen Schema abgearbeitet. So kann beispielsweise nur für einen Kunden eine Sonderanfertigung eines Produkts erstellt werden. Der generelle Prozess bleibt davon aber unberührt. Diese Klassifikation ist unabhängig von der in Abbildung 4 aufgezeigten.

## 3.2 Workflow Modification Language

In [5] wird eine Workflow Modification Language (WFML) definiert. Sie dient dazu, Änderungen von Workflowschemata zu beschreiben. Es wird eine (möglichst kleine) Menge an Primitiven definiert, die Modifikationen eines Workflows erlauben und dabei die Korrektheit sowohl für das Schema als auch für die Instanzen eines Workflows erhält. Bei diesem Ansatz werden Konzepte aus der Analyse der Schemaevolution objektorientierter Datenbanken wiederverwendet, da es sich nach Angabe der Autoren um ein ähnliches Problem handelt und deshalb die Lösung ebenfalls ähnlich sein sollte. Die Konsistenz wird in Struktur- und Verhaltenskonsistenz unterschieden. Bei der Strukturkonsistenz geht es um die statische Eigenschaften des Schema. Das geänderte Schema muss syntaktisch korrekt sein, das heißt es darf keine compile-time-errors enthalten. Bei der Verhaltenskonsistenz geht es um die Anpassung der laufenden Instanzen des Schemas. Sie fordert, dass die Anwendung (eines Primitives der WFML) auf einen Workflow diesen von einem Zustand in einen neuen legalen Zustand überführt. Ein Zustand ist legal, genau dann wenn die Ausführung ohne Laufzeitfehler fortgesetzt werden kann.
Sowohl die Verhaltens- als auch die Strukturkonsistenz werden durch alle Primitiven garantiert. Dadurch ist die Konsistenz auch für die Anwendung einer Sequenz von Primitiven gegeben. Casatis WFML teilt die Primitive wie folgt auf

- *Deklarationsprimitive* ändern die Deklaration der Workflowvariablen und -aktivitäten. *AddVar(VarName v, DefaultValue d)* fügt Variablen hinzu und ändert ihren Standardwert. *RemoveVar(VarName v)* entfernt Variablen. Variablen verschwinden implizit, wenn sie aufgrund der Flussstruktur nicht mehr benötigt werden. *AddTask(Task t)* fügt Aktivitäten hinzu.

- *Flussprimitive* ändern die Flussstruktur des Schemas. Sie modifizieren beispielsweise die

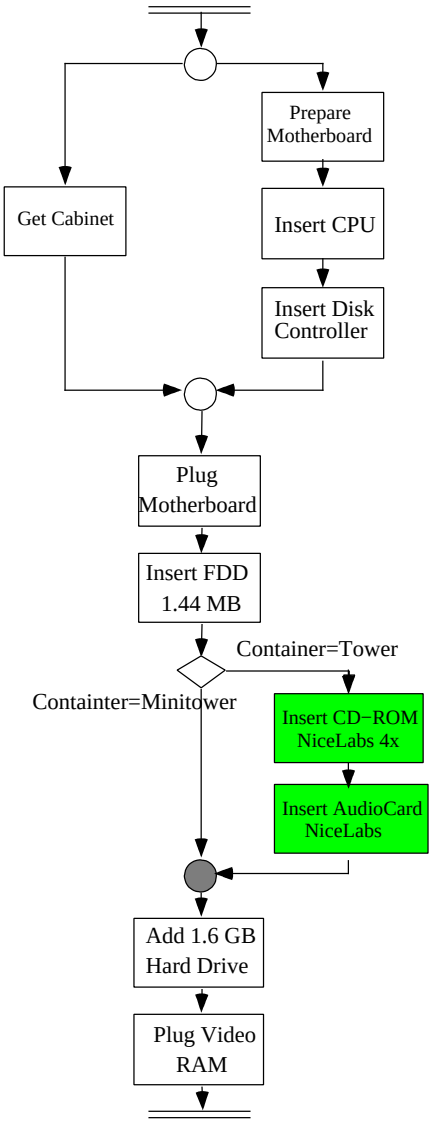


Abbildung 5: geändertes PC-Bau-WF-Schema. geänderte Tasks sind farbig hinterlegt.

Reihenfolge in der die einzelnen Aktivitäten abgearbeitet werden. *AddSuccessor(Task t, Task s)* fügt der Aktivität $t$ den Nachfolger $s$ hinzu. *RemoveSuccessor(Task t, Task s)* entfernt den Nachfolger $s$ aus der Menge der Nachfolger von Tasks $t$. *ModifyCondition(Condition c, Task t)* ändert die Startbedingungen der Aktivität $t$.

Die Modifikation in Abbildung 5 werden durch folgende Anweisungen erreicht:
*AddSuccessor(Insert_ Cd-rom_ BestCd_ 4x, Insert_ Cd-rom_ NiceLabs_ 4x)*
*AddSuccessor(Insert_ Cd-rom_ NiceLabs_ 4x, Insert_ AudioCars_ NiceLabs)*
*RemoveSuccessor(Insert_ FDD_ 1.44_ MB, Insert_ Cd-rom_ BestCd_ 4x)*

Die einzelnen Anweisungen einer WFML-Sequenz müssen atomar und in der gegebene Reihenfolge ausgeführt.
Die Modifikation des Schemas kann auch für aktive Prozesse durchgeführt werden. Läuft ge-

rade eine Aktivität, wird die Änderung des Schemas für diese Instanz nach Beendigung der Aktivität umgesetzt. Die nächste Aktivität wird entsprechend dem geänderten Schema ausgeführt. Offensichtlich führt das für *RemoveSuccessor(Insert_ FDD_ 1.44_ MB, Insert_ Cd-rom_ BestCd_ 4x)* genau dann zu einem Problem, wenn diese Aktivität gerade ausgeführt wird. In dem diesem Fall muss die Entscheidung, ob der Task beendet oder rückgängig gemacht wird, durch den Anwender getroffen werden.
Die Änderung des Workflows für den Bau des Rechners wird nur dann garantiert wirksam, wenn *Insert_ FDD_ 1.44_ MB, Insert_ Cd-rom_ BestCd_ 4x* noch nicht erfolgt. Für den Fall, dass es während der Migration erfolgt, kann wegen der Anwenderentscheidung keine generelle Aussage getroffen werden. Danach spielt die Migration offensichtlich keine Rolle mehr. Es wird deshalb der Begriff der „compliance", d.h. der Erfüllung eines Workflowschemas definiert. Dazu wird ein (initiales) Ausgangsschema $\Sigma^{W.I}$ und ein (finales) Zielschema $\Sigma^{W.I}$ benötigt.

**Satz 3.2.1.** *Compliance*
*Eine Instanz C des Schemas W.I erfüllt das Schema W.F, genau dann wenn*
$\forall t_n, t_n \in Completed_C^{W.I}$, *gilt* $\eta(\tau(\Sigma^{W.I}), \tau(S_C^{W.I}(t_n)), t_n) = \eta(\Sigma^{W.I}, S_C^{W.I}(t_n), t_n)$.

Die dabei genutzten Symbol bedeuten folgendes:

- $Completed_C^W$ bezeichnet die Menge der beendeten Aktivitäten der Instanz $C$.
- $S_C^W(t_n)$ ist der Status von $C$ nach Beendigung von $t_n$
- $\eta(\Sigma^W, S_C^W(t_n), t_n)$ beschreibt den Prozess der Ausführung von $C$.
- $\tau$ beschreibt die Anwendung eine WFML-Primitive.

Würde die Migration erst nach Beendigung von *Insert_ FDD_ 1.44_ MB, Insert_ Cd-rom_ BestCd_ 4x* versucht, entspricht $C$ nicht dem Schema $W.F$. In diesem Fall wird von einer bedingten Migration gesprochen, Um C dennoch nach W.F zu migrieren sind andere Modifikationen nötig (z.B. Rollback), die außerhalb der Möglichkeiten der WFML liegen. Im Falle der Erfüllung von $W.F$ durch $C$ wird von unbedingter Migration gesprochen.
Die Workflow Modification Language zeichnet sich dadurch aus, dass sie alle vorstellbaren Arbeitsabläufe definieren kann. So lässt sich aus einem Nullworkflow, einem Workflow ohne Elemente, jeder andere Workflow erstellen. Eher nachteilig ist, dass bei der Migration Nutzerentscheidungen erforderlich sind. Die „compliance" kann allerdings maschinell festgestellt werden und so Verstösse durch Nutzerentscheidungen gegen sie aufdecken. Dadurch kann der Anwender auf Fehler seinerseits hingewiesen.

## 3.3 Vererbung in Workflows

In [1, 2] unterscheidet man zunächst zwischen „Flexibilität durch Konfiguration" *(flexibility by configuration)* und „Flexibilität durch Adaption" *(flexibility by adaption)*. Ziel der Arbeiten sind vor allem die Reduzierung „echter" Änderungen, die einfache Verwaltung mehrerer Versionen bzw. Varianten und die Fehlervermeidung.

Bei der „Flexibilität durch Konfiguration"(FdK) handelt es sich nicht um eine Evolution in Sinne dieser Arbeit, da deren Hauptziel die Vermeidung von Änderungen ist. FdK erreicht sein Ziel durch eine Erweiterung der Möglichkeiten zur Definition der Flussstruktur. So wird beispielsweise die Unterstützung ein Funktion *SEQUENCE(<Menge von Aktivitäten>)* zur

Erweiterung der Definitionsmöglichkeiten von Workflows vorschlagen.
In bisherigen Systeme ist der Workflowmodellierer immer gezwungen eine explizite Reihenfolge innerhalb einer Sequenz festzulegen, auch dann wenn dies sachlich nicht geboten ist. *SEQUENCE(A,B,C)* ermöglicht es nun festzulegen, dass die Tasks A, B und C sequentiell auszuführen sind, legt aber deren Reihenfolge nicht fest. Stellt man später fest, dass es doch eine sinnvolle Reihenfolge gibt, ist man nicht mehr gezwungen eine bestehende Reihenfolge zu ändern, sondern löst lediglich die *SEQUENCE(A,B,C)* auf. Auf die Grenzen dieses Verfahrens wird hier nicht weiter eingegangen. Die Autoren weisen darauf hin, dass sich nicht alle möglichen Anpassungen durch dieses Verfahren abfangen lassen.

Bei „Flexibilität durch Adaption“ empfiehlt Aalst ein Vererbungskonzept. Dazu wurden die aus der objektorientierten Programmierung bekannten Methoden auf Workflows übertragen. Für die Informations- und die Operationssicht ist die Übertragung recht einfach. Klassen entsprechen den Workflowschemata und Objekte den Fällen bzw. Instanzen. Das klassische Vererbungsschema reduziert sich allerdings auf den statischen Aspekt. Nur Klassen können Vererbungsbeziehungen aufbauen. Ein Objekt kann seine Klasse nicht wechseln. Es behält sie vom Instantiieren bis zum Terminieren.
Es wird seitens der Autoren bemerkt, dass die Vererbung dynamischen Verhaltens auf der einen Seite nicht gut verstanden ist, auf der anderen Seite aber den wesentlichen Teil der Prozesssicht und des Prozessmanagements ausmacht.
Es werden vier Vererbungsbegriffe unterschieden. Im Folgenden sei x grundsätzlich Unterklasse von y. Von Protokollvererbung spricht man, wenn bei Ausführung aller Aktivitäten aus x, die auch in y vorkommen, es nicht möglich ist, das Verhalten von x und y zu unterscheiden. Sie fasst alle neuen Tasks zusammen und vergleicht deren sichtbares Verhalten. Die Routingmuster der Superklassen bleibt auch in den Unterklassen erhalten. Protokollvererbung „blockiert“ die neuen Aktivitäten.
Von Projektionsvererbung wird gesprochen, wenn bei Ausführung beliebiger Tasks aus x nur die Effekte der Tasks beachtet werden, die auch in y vorkommen. Projektionsvererbung versteckt die neuen Aktivitäten. Die Routingmuster der Superklasse bleiben erhalten. Da beide Vererbungsbegriffe orthogonal sind, führen die Autoren noch den Begriff der Protokoll/Projektionsvererbung ein. Von Life-Cycle-Vererbung wird gesprochen, wenn durch das Blockieren einiger neuer Aktivitäten in x und das Verstecken aller anderen neuen Aktivitäten das Verhalten von x und y nicht mehr unterscheidbar ist. Sie ist die allgemeinste Form der Vererbung. Die Vererbungsbegriffe klassifizieren Mengen von Workflows.

In Abbildung 6 sei N0 die Ausgangs- bzw Superklasse. N0 ist unter folgenden Vererbungsbegriffen Superklasse:

- *N1 - Protokoll- und Projektionsvererbung* Da sich N1 wie N0 verhält, wenn man D blockiert(Protokollvererbung). Ignoriert man nur die Ergebnisse von D, verhält es sich ebenfalls wie N0(Projektionsvererbung).

- *N2 - Protokollvererbung* Da sich N2 wie N0 verhält wenn man D blockiert, handelt es sich um Protokollvererbung. Lässt man D zu, verhält es sich anders wie N0. Projektionsvererbung ist dementsprechend nicht gegeben.

- *N3 - Projektionsvererbung* Es kann sich bei N3 nicht um Protokollvererbung handeln, da N3 nicht ausgeführt werden kann, wenn D geblockt wird. Wenn die Ergebnisse von D

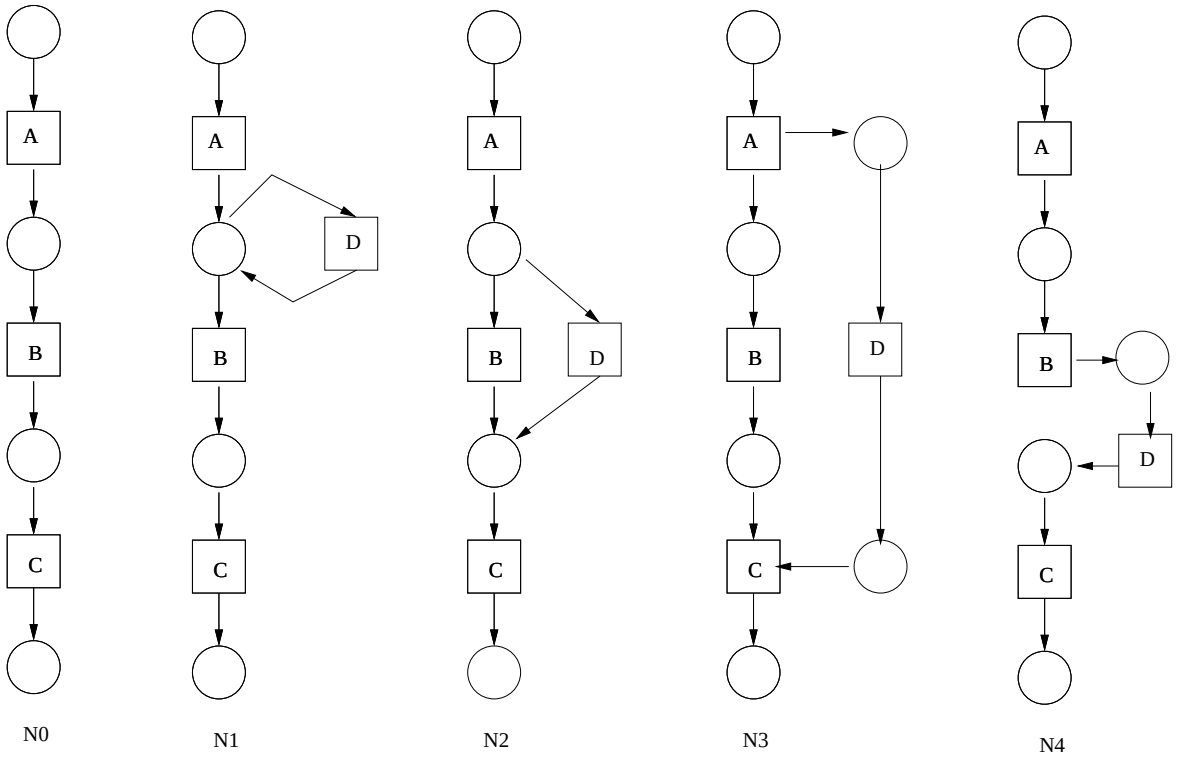


Abbildung 6: Workflowdefinitionen

lediglich ignoriert werden, kann N3 vollständig ausgeführt werden. Es verhält sich dann wie N0 unter Projektionsvererbung.

- *N4 - Projektionsvererbung* Die Begründung ist hier analog zu N3. Beim Blockieren von D kann N4 nicht beendet werden.

Außerdem stehen N1 bis N4 in einer Life-Cycle-Vererbungsrelation zu N0.

Die Konzepte hinter den Vererbungsbegriffen machen lediglich eine Aussage über die Beziehung der Schemata zueinander. Sie haben keine konstruierenden Funktion.
Deshalb werden auf Basis der verschiedenen Vererbungsbegriffe die folgenden Transformationsregeln definiert.

- Die Transformationsregel PT erhält die Protokoll und life-cycle-Vererbung. PT erweitert die Superklasse um neue Alternativen. In der erzeugten Unterklasse gibt es alternative Routen mit neuen Aktivitäten.
- Die Transformationsregel PP bewahrt alle 4 Vererbungsformen. PP fügt neue Aktivitäten hinzu, die nur das Verhalten aufschieben.
- Die Transformationsregel PJ erhält die Projektionsvererbung und life-cycle-Vererbung. PJ fügt neue Task zwischen bestehende ein. Die eingefügten Aktivitäten können einfacher Natur sein oder zusammengesetzt, d.h. vollständige Unterworkflows, sein.
- Die Transformationsregel PJ3 erhält wie PJ die Projektionsvererbung und die life-cycle-Vererbung und fügt parallel Verhalten hinzu.

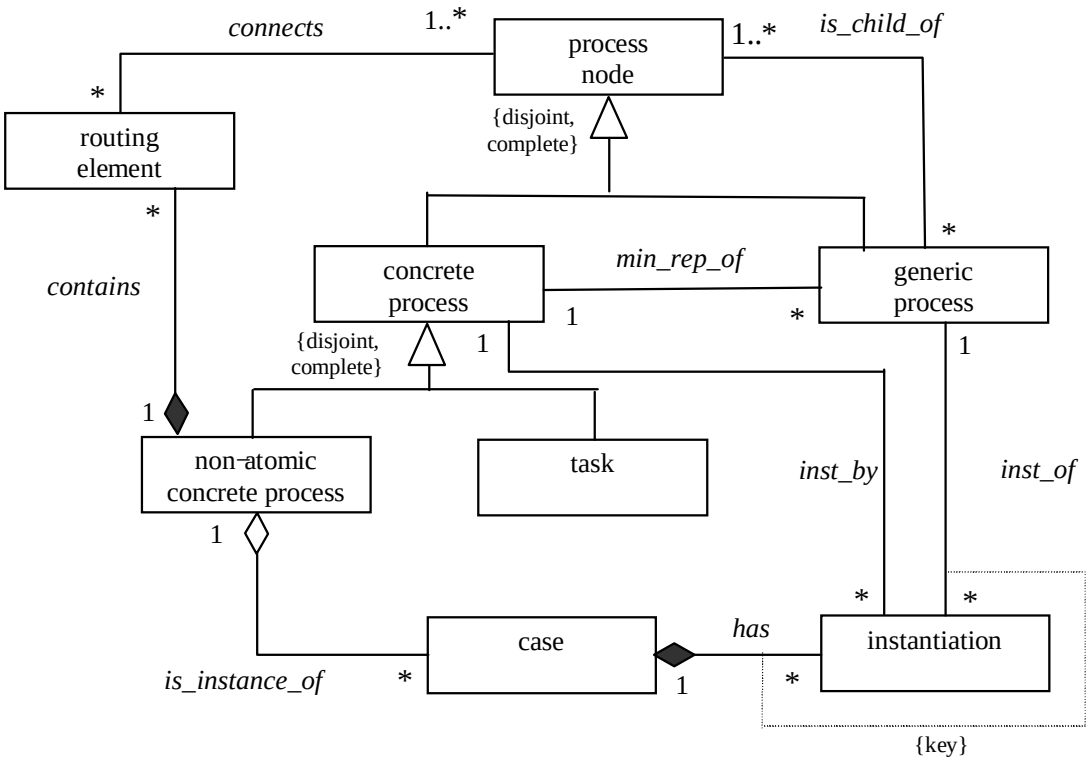


Abbildung 7: Klassenmodell für das Konzept des minimalen Repräsentanten

Die Transformationsregeln passen zu oft benutzten Designkonstrukten. Es handelt sich um die Auswahl, die sequentielle Komposition und die parallele Komposition. Unter Beachtung der Regeln ist Vererbung garantiert. Außerdem kann jeder Workflow ohne Laufzeitprobleme (wie z.B. Deadlock oder unnötige mehrfaches Ausführung) konvertiert werden. Die Vererbung beschränkt die Änderung und eignet sich für momentane und evolutionäre Änderungen. Sie ist geeignet, um Erweiterungen vorzunehmen und vermeidet transiente Fehler. Gerade wegen der Möglichkeit Fehler in geänderten Workflows zu machen, bevorzugt Aalst grundsätzlich „Flexibilität durch Konfiguration“.

## 3.4 Minimaler Repräsentant

Wie bereits dargestellt, besteht das Hauptproblem der Evolution in der Transformation laufender Prozesse von einem in ein anderes Schema. Dazu wird in [3] der Begriff des minimalen Repräsentanten eingeführt. Der Aufsatz abstrahiert von der reine Evolution von Prozessen und sucht nach Gemeinsamkeiten. Durch diesen Ansatz können auch Informationen für die Führung eines Unternehmens aus den unterschiedlichen Prozessen zusammengefasst werden.

Wie in Abbildung 7 zu sehen, können Prozessknoten entweder ein konkreter oder ein generischer Prozess sein. Über die generischen Prozesse lässt sich eine beliebig tiefe Kette von Prozessknoten bauen. An deren Ende und nur da befindet sich ein konkreter Prozess. Dieser konkrete Prozess stellt den minimalen Repräsentanten aller generischer Prozesse der Kette dar. Ein generischer Prozess kann beliebig viele Kinder haben. Dies ermöglicht den Aufbau von Prozesshierarchien. Konkrete Prozess sind ihrerseits entweder Aktivitäten oder nicht-atomare bzw. zusammengesetzte Prozesse. Die zusammengesetzten Prozess bilden dann die Grundlage

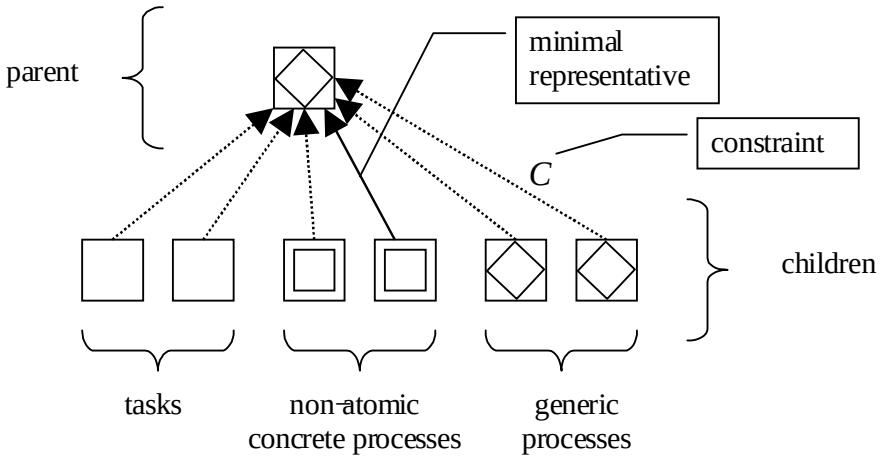


Abbildung 8: Symbole in Vererbungsdiagrammen

für die Fälle. Die Fälle sind zugleich Instanzen der generischen Prozesse. Generische Prozesse lassen sich nicht instantiieren. Für die einzelnen Elemente gelten Beschränkungen, auf die hier nicht weiter im Detail eingegangen werden soll. Dazu gehört zum Beispiel, dass die Beziehungen zwischen den generischen Prozessen azyklische sein müssen.
Routingelement wie Verzweigung und Vereinigung stehen etwas abseits. Sie sind zwar Teil konkreter Prozesse, aber nicht hierarchisch gliederbar.

Das Klassendiagramm zeigt 3 Arten von Informationen.

1. *Routinginformationen* beschreiben nicht-atomare konkrete Prozesse. Sie spezifizieren, welche Aktivitäten, nicht-atomare konkrete Prozesse und generische Prozesse genutzt werden und in welcher Reihenfolge dies geschieht.

2. *Vererbungsinformationen* beschreiben die Beziehungen zwischen generischen Prozessen und seinen Kindern. Sie legen mögliche Instantiierungen generischer Prozesse durch konkrete Prozesse fest und betreffen neben den beiden Prozessarten auch Prozessknoten, die *is_ child_ of* und die *min_ rep_ of* Beziehung.

3. *dynamische Informationen* umfassen die Ausführung der Fälle und die Instantiierung der generischen Prozesse durch konkrete Prozesse. Sie involviert die *is_ instance_ of*-, die *has*-, die *inst_ by* und die *inst_ of*-Beziehung.

Die letzten beiden Informationsarten unterscheiden sich durch die beschriebenen Beziehungen.

Zur Darstellung der Idee des minimalen Repräsentanten werden zwei Diagrammarten definiert. Das Routingdiagramm definiert für nicht-atomare konkrete Prozesse das Routing der Fälle entlang der Prozessknoten. Außerdem werden generische Prozesse in diesen Diagrammtyp eingegliedert.
Abbildung 8 zeigt ein Vererbungsdiagramm. Die Wurzel eines solchen Diagramms ist ein generischer Prozess, der als *parent* bezeichnet wird. Alle anderen Knoten sind dementsprechend Kinder. Jeder nicht-atomare konkrete Prozess referenziert auf ein Routingdiagramm. Jeder generische Kind-Prozess referenziert wiederum auf ein Vererbungsdiagramm. Jeder generische Prozesse hat, wie bereits geschrieben, einen minimalen Repräsentanten. Diese Beziehung wird im Diagramm durch eine durchgezogene Linie markiert. Alle anderen Beziehungen werden

durch gestrichelte Linien dargestellt. Der minimalen Repräsentant kann als Superklasse im objektorientierten Sinne betrachtet werden. Es entstehen Prozessfamilien.

Die wesentlich Idee des Konzept ist, dass der Wechsel nur zwischen Kindern eines generischen Prozesses also Mitgliedern der selben Prozessfamilie erfolgt. Das reduziert die Möglichkeiten möglicher Transformationen und verbessert so deren Handhabbarkeit. Würde man Transformationen von $n$ Mitgliedern einer Prozessfamilie direkt auf jedes andere Familienmitglied definieren, erhielte man $n(n-1)$ Mappings. An dieser Stelle kommt nun der minimalen Repräsentant zum Tragen. Jede Transformation erfolgt nun über diesen Repräsentanten. Dadurch müssen nur noch $2(n-1)$ Mappings von den Familienmitgliedern zum minimalen Repräsentanten definiert werden. Kommt ein neues Mitglied hinzu, müssen nun nur noch zwei Mappings definiert werden – vom neuen Mitglied auf den minimalen Repräsentanten und umgekehrt.

Die Idee hat allerdings auch Nachteile. Die Möglichkeiten der Transformation hängen von den Möglichkeiten des minimalen Repräsentanten ab Informationen aufzunehmen. Sind diese sehr beschränkt, sind auch die Transformationsmöglichkeiten entsprechend eingeengt. Neue Probleme können zu neuen minimalen Repräsentanten führen und zu entsprechendem Aufwand bezüglich der Mappinginformationen.

Die Transformation zum minimalen Repräsentanten hin wird durch eine Generalisierungsfunktion definiert. Der umgekehrte Weg wird durch eine Spezialisierungsfunktion festgelegt. Alle Kinder haben einen Zustandsraum. Die Generalisierungsfunktion ist partiell und definiert für einige Zustände eines Kinds einen Übergang zum minimalen Repräsentanten. Diese Zustände werden als „jump states“ bezeichnet. Alle anderen Zustände werden als „wait states“ (Wartezustände) bezeichnet. Der Übergang hin zum minimalen Repräsentanten ist nur in den „jump states“ möglich. Befindet sich ein Fall in einem „wait state“, wird die Transformation solange aufgeschoben bis ein „jump state“ erreicht ist. Für diese Zeit wird der Fall entsprechend dem alten Schema abgearbeitet.

Abschließend sei noch gesagt, dass dieses Konzept zum Zeitpunkt des Erscheinens von [3] in keinem Produkt verwirklicht ist.

# 4 Abbildung von Schemaevolution in standardisierten WfM-Sprachen

Nachdem im letzten Abschnitt einige theoretische Ansätze zu Schemaevolution vorgestellt wurden, soll in diesem Abschnitt auf mögliche Umsetzungen der Ideen und Vorschläge eingegangen werden. Es werden Standards betrachtet, insbesondere formalen Sprachen und Austauschformate und deren Möglichkeiten Workflowevolution zu unterstützen. Standards bilden ihrerseits häufig eine Grundlage für die Modelle in Produkten und Werkzeugen, die im nachfolgende Abschnitt behandelt werden. Es soll die praktische Seite der Workflowevolution betrachtet werden.

Prozesse lassen sich blockstrukturiert und auf Basis von gerichteten Graphen definieren. Während gerichtete Graphen sich eher mit graphischer Notation verbinden lassen und betriebswirtschaftlich ausgebildeten Personen eher entgegen kommen, befinden sich blockstrukturierte Sprachen näher an den üblichen imperativen Programmiersprachen. Blockstrukturierte Sprachen lassen sich relativ leicht in graphorientierte Sprachen umwandeln. Die umgekehrte Richtung ist schwieriger. Graphische Anwendungsprogramme, die eine blockorientierte Sprache (z.B. BPML) als Zielformat haben, schränken deshalb die Möglichkeiten der Kanten des Graphen ein.

## 4.1 Business Process Modeling Language

BPML ist eine Metasprache zur Beschreibung von Geschäftsprozessmodellen. Man kann sie sich als blockorientierte Programmiersprache vorstellen. Rekursive Blockstrukturen spielen eine wesentliche Rolle bei der Definition und Ausführung der Prozesse. Sie steuern den Ablauf (engl.: Routing). BPML ist als Implementierungssprache konzipiert. Es unterstützt rekursive bzw. hierarchische Prozessdefinitionen.
Die Business Process Modeling Language konzentriert sich auf die Definition von Webservices. Dies drückt sich durch folgende Punkte aus.

- Spezifische Aktivitätstypen für Nachrichtenaustausch, Event-behandlung, Ausgleich (compensation (in case of failure)) und Verzögerung.
- Attribute zur Unterstützung von Instanzkorrelation, zum Extrahieren von Nachrichtenteilen und zum lokalisieren bzw. orten von Serviceinstanzen
- Unterstützung von Transaktionen, Nutzung von Blockstrukturkontext, Ausnahmebehandlung

## 4.2 Web Services Business Process Execution Language

An der Web Services Business Process Execution Language(WS-BPEL), vorher als BPEL4WS bekannt, wird innerhalb der OASIS zur Zeit an der Version 2.0 gearbeitet. WS-BPEL ist eine formale Sprache zur Spezifikation von Geschäftsprozessen und „business interaction protocols“. Es ist spezialisiert auf den Austausch von Daten und Dokumenten über Webservices und unterstützt so primär B2B-Transaktionen. Sie ist eine Untermenge von BPML und für den allgemeinen Einsatz als Business Process Execution Language eher weniger geeignet. Zur Versionierung von Geschäftsprozessen äußert sich der Standard überhaupt nicht. BPEL4WS ist im Jahr 2002 von IBM und Microsoft entwickelt worden. Die ihrerseits vorher die Sprachen WSFL (IBM) und XLANG (Microsoft) propagierten.
BPEL ist eine blockstrukturierte Programmiersprache, die den rekursiven Aufruf von Blöcken ermöglicht. Die Definition und Deklaration geschieht aber nur auf der Ebene der Prozesse und ist dementsprechend nicht rekursiv. Dies ist vergleichbar mit der Möglichkeit, Funktionen nur in einer Ebene (wie z.B. in C) definieren zu können. Innerhalb eines Blocks werden graphbasierte Konzepte zur Flussdefinition unterstützt. Diese sind auf azyklische Graphen beschränkt. Sie erlauben also keine Schleifen. BPEL unterstützt keine Unterprozesse.
BPEL baut auf anderen Webtechnologien auf, wie z.B. WSDL 1.1, XML Schema 1.0, XPath 1.0, und WS Addressing. In den BPEL-Dokumenten wird der Prozessablauf beschrieben. Er bezieht sich dabei auf in WSDL-Dateien beschriebene Webservices. WSDL-dateien beschreiben die Struktur der Nachrichten(Datencontainer), Operation und Porttypen. Sie binden auch die Porttypen an ein festgelegtes Protokoll (z.B. SOAP). Diese Bindungen sind ihrerseits mit speziellen URIs verbunden, die den Webservice anbieten.

## 4.3 Business Process Modeling Notation

Business Process Modeling Notation(BPMN) [18] ist ein Standard der Object Management Group. Das Hauptziel besteht in der graphischen Notation von Geschäftsprozessen mittels Business Process Diagram (BPD). Es soll die Kommunikation zwischen Anwendern und Entwicklern erleichtern und so die Lücke zwischen Prozessdesign und deren Implementierung

schließen.
Dadurch soll eine schnellere Implementierung von Geschäftsprozessen erreicht werden. Aus organisatorische Sicht wird dadurch die Änderungen von Prozessen unterstützt. Diagramme können mit einem Versionsattribut versehen werden.
Der Standard enthält ein Kapitel zur Abbildung von BPMN auf BPEL. Auf der Homepage des Standards [18] ist ein Vergleich mit UML 2.0 Activity Charts zu finden.

## 4.4 XML Process Definition Language

Die XML Process Definition Language(XPDL) [16] wurde durch die Workflow Management Coalition verabschiedet. Die Sprache ist, wie der Name bereits sagt, xml-basiert und dient dem Austausch von Geschäftsprozessdefinitionen. Sie ist als Dateiformat konzipiert und stellt ein herstellerunabhängiges Austauschformat dar. Die Sprache enthält die Möglichkeit die Paket- und Prozessdefinitionen mit einem Versionsattribut zu versehen. Pakete sind eine Menge von Prozessen. Die aktuelle Version 2 wurde um Elemente erweitert, die den Austausch von BPMN-Modellen zu unterstützen.
Sie ist eine graphorientierte Sprache, unterstützt aber auch Blöcke. Verschachtelte Prozesse sind nicht möglich. Das Routing wird durch Übergänge (Transitionen) gesteuert. Diese Übergänge können mit Bedingungen versehen werden und bilden die Kanten des Graphen. Die Aktivitäten sind Knoten des gerichteten Graphen.
XPDL legt sein Hauptaugenmerk auf die Verteilung der Arbeit.

- Es definiert durch Aktivitätsattribute die Resourcen, die zur Ausführung benötigt werden. Bestimmt wird dies durch einen Ausdruck, der zur Laufzeit ausgewertet wird.
- Aktivitätsattribute spezifizieren die Applikation, welche zur Ausführung der Aktivität benötigt werden.
- Beide genannten Punkt unterstützen die Notation der Ressourcen in Verbindung mit der Anwendungssoftware zur Ausführung der Aktivitäten.

## 4.5 Yet Another Workflow Language (YAWL)

YAWL [4] wurde durch Professor Aalst und seine Mitarbeiter an der Universität Eindhoven entwickelt, stammt also nicht von einem Standardisierungsgremium. Es basiert auf einer Analyse der bestehende Workflowmanagementsysteme und -standards und nutzt eine reichhaltige Menge an Workflowmustern. In Tabelle 1 ist eine Übersicht [20] zu sehen, die die Unterstützung einzelner Workflowmuster durch verschiedene Standards beschreibt. Die Analyse zeigte, dass sowohl die Standards (zum Zeitpunkt der Analyse) als auch die theoretischen Modelle wie z.B. Petrinetze nur eingeschränkt nutzbar für die Darstellung der wesentlichen Muster sind.
Es werden Petrinetze als Ausgangspunkt genutzt, um der Sprache ein sicheres theoretisches Fundament zu geben. Das Fehlen einer derartigen Basis bei bestehenden Sprachen wie XPDL wird mit als Grund für die Entwicklung der YAWL genannt. Petrinetze bilden zwar den Ausgangspunkt. Das Ergebnis der Nutzung von YAWL sind allerdings keine. Es handelt sich um Transitionssysteme.
YAWL hat eine formale Semantik und bietet eine graphische Repräsentation bzw. Notation. In Abbildung 9 sind die Symbole aufgeführt. Die Autoren glauben, dass sich die Sprache wegen

| Muster | Standard | | | | |
|---|---|---|---|---|---|
| | BPEL | XLANG | WSFL | BPML | WSCI |
| Sequence | + | + | + | + | + |
| Parallel Split | + | + | + | + | + |
| Synchronization | + | + | + | + | + |
| Exclusive Choice | + | + | + | + | + |
| Simple Merge | + | + | + | + | + |
| Multi Choice | + | - | + | - | - |
| Synchronizing Merge | + | - | + | - | - |
| Multi Merge | - | - | - | +/- | +/- |
| Discriminator | - | - | - | - | - |
| Arbitrary Cycles | - | - | - | - | - |
| Implicit Termination | + | - | + | + | + |
| MI without Synchronization | + | + | + | + | + |
| MI with a Priori Design Time Knowledge | + | + | + | + | + |
| MI with a Priori Runtime Knowledge | - | - | - | - | - |
| MI without a Priori Runtime Knowledge | - | - | - | - | - |
| Deferred Choice | + | + | - | + | + |
| Interleaved Parallel Routing | +/- | - | - | - | - |
| Milestone | - | - | - | - | - |
| Cancel Activity | + | + | + | + | + |
| Cancel Case | + | + | + | + | + |

Tabelle 1: Übersicht zu musterunterstützenden Standards

ihrer Ausdrucksstärke und ihrer formalen Semantik als Zwischensprache für die Übersetzung anderer Workflowsprachen anbietet.

Bei YAWL besteht ein Workflow aus einer Menge von Prozessen. Ein Prozess setzt sich aus Tasks zusammen. Diese werden in atomare und zusammengesetzte Task (engl.: composite tasks) unterteilt. Zusammengesetzte Tasks verweisen wiederum auf eine Prozessdefinition. Dadurch lässt sich eine hierarchische Struktur abbilden. Es gibt genau einen Wurzelprozess (top level process) innerhalb eines Workflows. Auf diesen darf bzw. wird durch keinen zusammengesetzten Task verwiesen. Er bildet die Wurzel der hierarchischen Prozessdefinition. Im Unterschied zu Petrinetzen können sich in einem mittels YAWL definierten Workflow mehrere Transitionen, d.h. Task hintereinander befinden. Es müssen nicht immer Aus- und Eingangsbedingungen definiert werden.

YAWL hat eine XML-Syntax. Es liegt ein Schema vor. Es wurde ein Prototyp für die Ausführung von Workflows entwickelt. Außerdem ist ein Workflow-Designer entwickelt worden. Die Abbildung 10 gibt einen kurzen Überblick über die Architektur den Systems. Die Werkzeuge sind nicht noch einmal im Kapitel 5 aufgeführt.

# 5 Schemaevolution in WfMS

In diesem Abschnitt soll für einige ausgewählte Werkzeuge darüber Auskunft gegeben werden, inwieweit sie die Workflowevolution unterstützen.

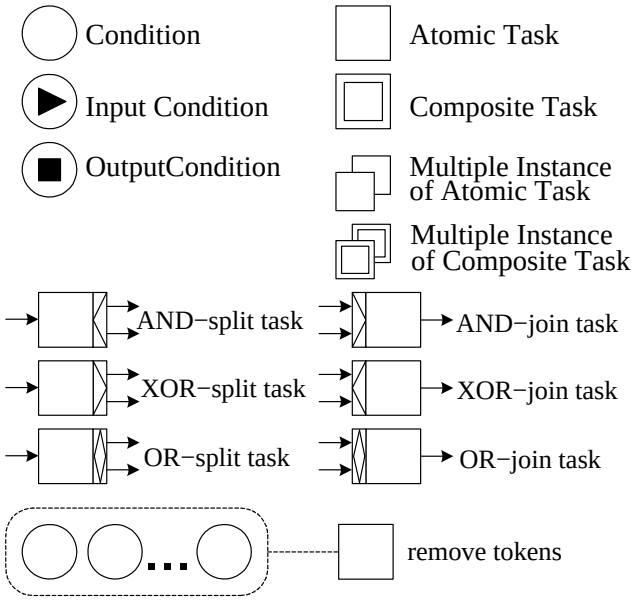


Abbildung 9: YAWL-Symbole

Fujitsu's Interstage Business Process Manager, vorher unter dem Namen I-Flow bekannt (davor als TeamWare Flow), basiert auf Java und ist eine kommerzielle Workflow-engine. Es bietet Prozessmodifikation zur Laufzeit der Prozesse durch das Hinzufügen von Unterprozessen. I-Flow enthält Internet Inter-ORB (IIOP) protokollbasierte Komponenten und kann externe (Legacy) Software durch CORBA ansprechen.

JBoss jBPM [22] bezeichnet sich als flexibles, erweiterbares Workflowmanagementsystem. Es ist Open Source und java-basiert. Außerdem erhebt es den Anspruch das verbreitetste System zu sein. In der aktuellen Version (3.0) unterstützt die Software die Versionierung von Workflowdefinitionen. Neue Instanzen werden immer auf Basis der neuesten Version gebildet. Bestehende Instanzen können nicht auf eine neuere Version migriert werden. Sie laufen bis zum Abschluss in der Version in der sie gestartete wurden weiter. Es kann die Änderungsstrategie Flush genutzt werden.

ActiveBPEL [21] ist eine Laufzeitumgebung zum Ausführen von Prozessen. Wie der Name bereits andeutet, müssen diese in BPEL (siehe 4.2) definiert sein. Hauptziel des Programm ist die vollständige Unterstützung des Standards.

con:cern [23] ist eine Workflow-engine, die nicht auf einer der Standardsprachen bzw. -modelle aufbaut, sondern einen eigene Ansatz verfolgt. Ein Prozess ist als Menge von Aktivität mit Vor- und Nachbedingungen definiert. Eine Aktivität wird ausgeführt wenn die Vorbedingungen erfüllt sind, d.h. es erfolgt in Abweichung zu den üblichen Modellen keine explizite Festlegung der Reihenfolge der Aktivität. Durch die Durchführung der Aktivität werden Nachbedingungen erzeugt.
Die Entwickler der Software glauben, das dieses Konzept bisherigen Ansätzen überlegen ist, wenn mindestens einer der folgenden Punkte zutrifft:

- komplexe Prozesse mit Ausnahmen und Spezialfällen
- Abhängigkeit der Aktivitätenreihenfolge von mehreren Faktoren

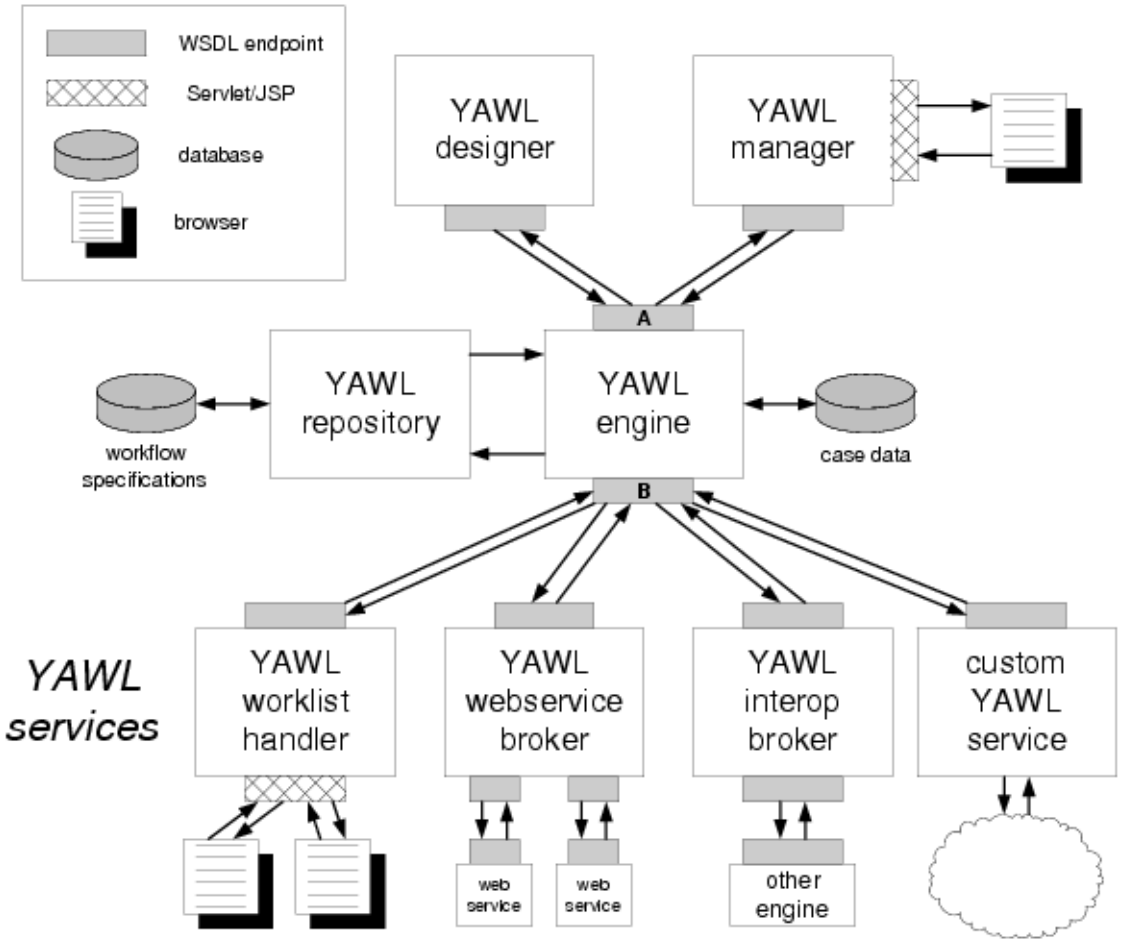


Abbildung 10: YAWL-Architektur

- die Möglichkeit manuell in den Prozessablauf einzugreifen
- inhaltsbasierte Abhängigkeiten zwischen den Aktivitäten
- strenge/starke Anforderungen an Modularität und Flexibilität
- lose Prozesskopplung

con:cern erlaubt den direkten Eingriff in laufende Prozesse und das schrittweise Anpassen der Prozesse an neue Erfordernisse. Damit verfügt die Software über eine der weitestgehensten Schemaevolutionsmöglichkeiten. Inwieweit alle Vorbedingungen vom Anwender bereits vorhergesehen werden können und Änderungen der Prozesse somit fehlerfrei möglich sind, wird hier nicht betrachtet. con:cern ist Open Source (LGPL) und läuft nach eigenen Angaben auf jedem J2EE-kompatiblen Applikationsserver.

Shark [24] ist ein erweiterbares Workflow-engine-Framework, deren Implementierung auf dem WfMC-Standard XPDL basiert. Die Verbindung zu den Aktivität, genauer deren Applikationen erfolgt über die WfMC „ToolAgents“ API. Prozesse lassen sich über Enhydra JaWE, einen grafischen XPDL-Editor entwerfen.

WfMOpen [25] ist eine J2EE-basierte Implementierung eine Workflow-engine auf Basis der Standards der Workflow Management Coalition und der Object Management Group. Auf Basis der OMG Workflow Management Facility Specification V1.2 wurde ein Interface entwickelt, das auch den Anschluss von Diensten über CORBA an den Kern des WfMS ermöglicht. Workflows werden mittels der (um einige Erweiterungen ergänzten) XML Process Definition Language spezifiziert. WfMOpen liefert Komponenten zur Ansteuerung von Webservice mittels SOAP.

OpenWFE [26] ist ein Workflowmanagementsystem bestehend aus einer Workflow-Engine, einem webbasierten Prozessdesigner(Droflo) und Komponenten zur Ansteuerung automatischer Agenten(APRE). Automatischen Agent sind in diesem Zusammenhang Applikationen, die bei erreichen einer Aktivität automatisch durch das WfMS gestartet werden. Das System verfügt über die übliche Funktionalität laufende Prozesse zu Überwachen und den Nutzern ihre Aufgabe mitzuteilen. Es steht unter einer BSD Lizenz und ist in Java implementiert, bietet aber Schnittstellen(REST) zu anderen Sprachen und Umgebung (C#, Python, Perl). Prozessdefinitionen werden durch eine eigene, erweiterbare, xml-basierte Sprache beschrieben. Die Dokumentation geht auf die Implementierung der von Prof. Aalst entwickelten Workflowmuster [20] ein.

Leider muss festgestellt werden, dass die Unterstützung für eine vollständige Schemaevolution bisher in keinem Tool gegeben ist. Zwar ist sind hier und da ein paar Ansätze zu erkennen, realisiert ist aber keines der Konzepte. Darüber hinaus ist festzustellen, dass wenn überhaupt nur durch spezifische Erweiterungen der Standards Möglichkeiten zur Schemaevolution eröffnet werden.

# 6 Zusammenfassung

Diese Arbeit betrachtete verschiedene Ansätze zur Definition von Workflows und die daraus entwickelten Möglichkeiten zur Workflowevolution. Außerdem wurden Standards verschiedener Gremien und einige Werkzeuge bezüglich ihrer Fähigkeiten zur Unterstützung der Evolution untersucht.
Die Forschungsergebnisse zeigen, dass die Möglichkeiten der automatisierten Unterstützung von Workflowevolution begrenzt sind. Die Grenze selbst ist allerdings noch nicht genau festgelegt. So überschreitet der Ansatz der Workflow Modelling Language diese Grenze und erfordert desöfteren Nutzereingriffe. Die Evolution auf Basis von Vererbung und die daraus abgeleiteten Transformationsregeln erfordern zwar keine direkten Nutzereingriffe, begrenzen aber die Möglichkeiten der Evolution. Bei den Standards und Werkzeugen kann das Fazit gezogen werden, dass noch einiges zu tun ist, bis die Möglichkeiten der bereits erforschten Evolutionsansätze umgesetzt sind.

# Literatur

[1] W.M.P. van der Aalst, T. Basten: Inheritance of workflows: an approach to tackling problems related to change. Theoretical Computer Science, 2002

[2] W.M.P. van der Aalst, S. Jablonski: Dealing with workflow change: Identification of issues and solutions. Int. J. Computer Syst., Science and Engineering 15(5), 267-276 (2000)

[3] W.M.P. van der Aalst.: Generic Workflow Models: How to Handle Dynamic Change and Capture Management Information? coopis, p. 115, Fourth IECIS International Conference on Cooperative Information Systems, 1999

[4] W.M.P. van der Aalst, L. Aldred, M. Dumas, A.H.M. ter Hofstedei :Design and implementation of the YAWL system, http://is.tm.tue.nl/research/patterns/download/yawls_long_qutrep.pdf

[5] F. Casati, S. Ceri, B. Pernici, G. Pozzi: Workflow evolution. Data and Knowledge Engineering 24(3), 211-238 (1998)

[6] S. Rinderle, P. Dadam: Schemaevolution in Workflow-Management-Systemen. Informatik-Spektrum, Volume 26, Issue 1, Jan 2003, Pages 17 - 19

[7] S. Rinderle, M. Reichert, P. Dadam: Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata. Informatik - Forschung und Entwicklung, Volume 17, Issue 4, Dec 2002, Pages 177 - 197

[8] M. Reichert, P. Dadam: $\text{ADEPT}_{\text{flex}}$ - Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, 10(2):93-129, March / April 1998 http://www.informatik.uni-ulm.de/dbis/01/staff/reichert/papers/journals/reda98c.pdf

[9] M. Kradolfer, A. Geppert: Dynamic Workflow Schema Evolution based on Workflow Type Versioning and Workflow Migration. Proc. of the 4th International Conference on Cooperative Information Systems, 1999

[10] R. Müller, E. Rahm: Dealing with Logical Failures for Collaborating Workflows, Proceedings of Fifth International Conference on Cooperative Information Systems (CoopIS), Eilat, Israel, Sep. 2000. Springer, Berlin, 2000: 210-223 http://lips.informatik.uni-leipzig.de/pub/2000-40

[11] R. Müller, E. Rahm: Workflow-Managment-System, Vorlesung Sommersemster 2003

[12] G. Vossen, J. Becker: Geschäftsprozeßmodellierung und Workflow-Management: Modell, Methoden, Werkzeuge, International Thomson Publishing 1996, ISBN 3-8266-0124-6

[13] Workflow Management Coalition Terminology & Glossary, Februar 1999 http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf

[14] BPML | BPEL4WS, A Convergence Path toward a Standard BPM Stack, August 2002, BPMI.org http://www.bpmi.org/downloads/BPML-BPEL4WS.pdf

[15] Workflow & BPM Research http://www.workflow-research.de/Links/

[16] Workflow Process Definition Interface – XML Process Definition Language, Version 2.0, Document Number WFMC-TC-1025, Oct 2005 http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf

[17] R. Shapiro: A Comparison of XPDL and BPML and BPEL: Cape Visions, 2002 http://www.ebpml.org/A_Comparison_of_XPDL_and_BPML_BPEL.doc

[18] Business Process Modeling Notation `http://www.bpmn.org`

[19] Workflow Management Coalition `http://www.wfmc.org`

[20] Workflow Patterns `http://www.workflowpatterns.com`

[21] ActiveBPEL `http://www.activebpel.org`

[22] JBoss jBPM `http://www.jboss.com/products/jbpm`

[23] con:cern `http://con-cern.org/`

[24] Enhydra Shark `http://shark.objectweb.org/`

[25] WfMOpen `http://wfmopen.sourceforge.net`

[26] OpenWFE `http://web.openwfe.org/display/openwfe/Home`