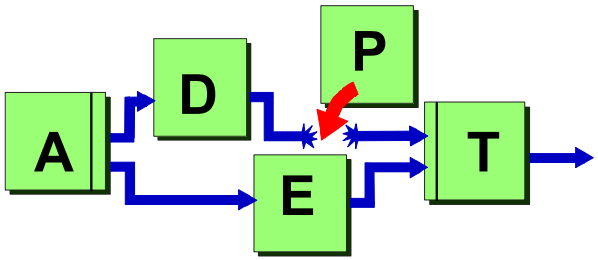
Das ADEPT-Projekt

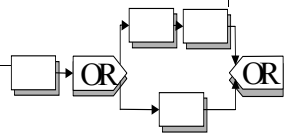
(Reichert & Dadam, Universität Ulm)

- ADEPT = Application Development Based on Encapsulated Pre-Modeled Process Templates



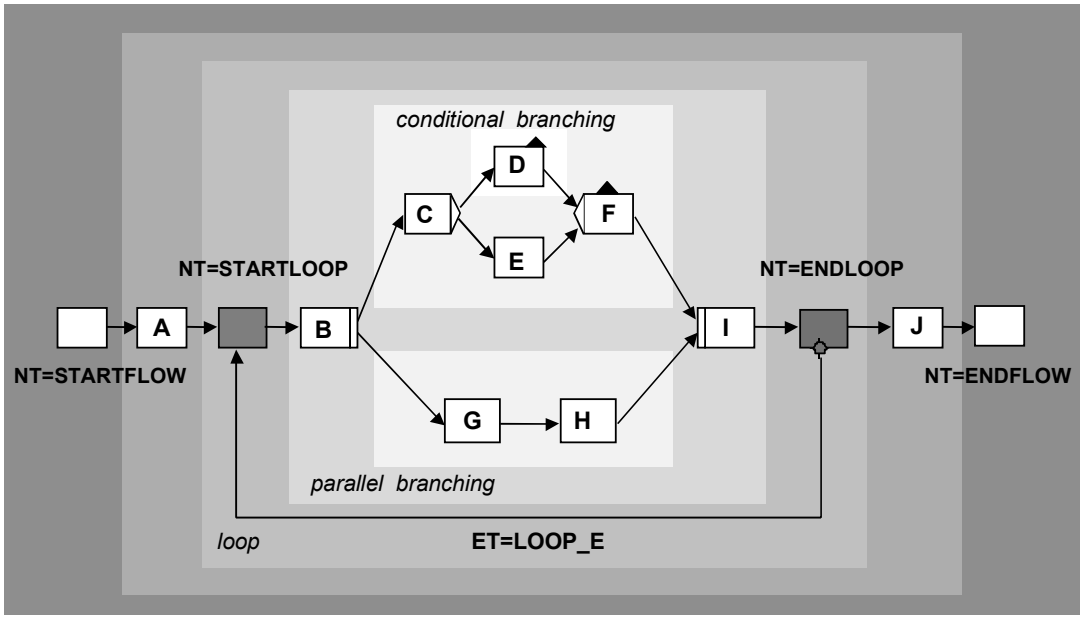
http://www.informatik.uni-ulm.de/dbis/index.html

- Kernziele:
  - Unterstützung der Entwicklung von robusten, transaktionalen und prozessorientierten Informationssystemen ($ADEPT_{BASE}$)
  - Konzepte, Architektur und Implementierung von adaptiven Workflow-Systemen ($ADEPT_{FLEX}$)
  - Unterstützung von unternehmensweiten Workflow-Applikationen



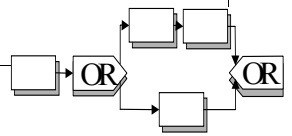
$ADEPT_{base}$: Workflow-Definitionen

- Block-orientierte Kontrollflußorganisation
  - Hierarchisch und symmetrisch organisiert; keine nicht-hierarchischen Überlappungen
  - Wohldefinierte Start- und Endknoten



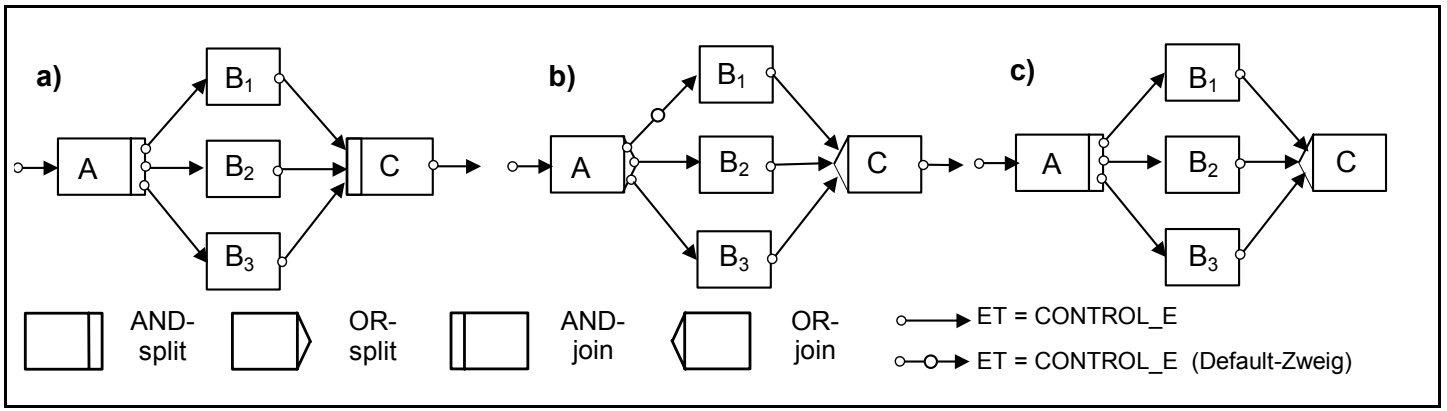
NT: Node Type
ET: Edge Type

- Verschiedene Modi der Parallelausführung
- Synchronisationskanten

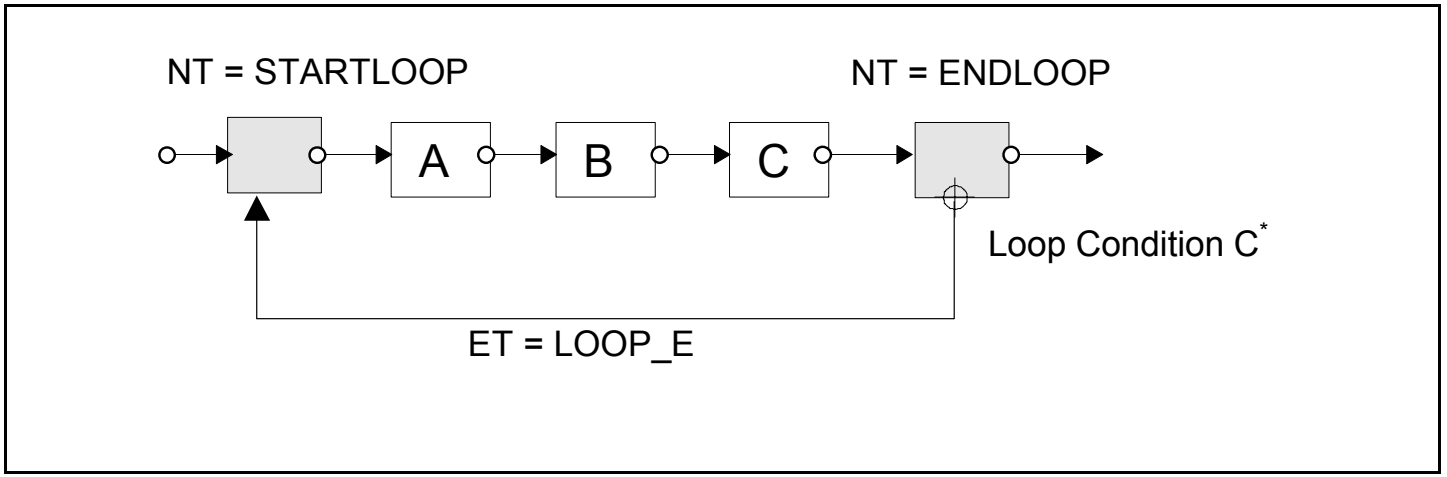


ADEPT$_{base}$: Kontrollfluß (1)

- Und/Oder-Verzweigungen:
  - Parallele Verzweigung (a)
  - Bedingte Verzweigung (b)
  - Parallele Verzweigung mit finaler Auswahl (c)



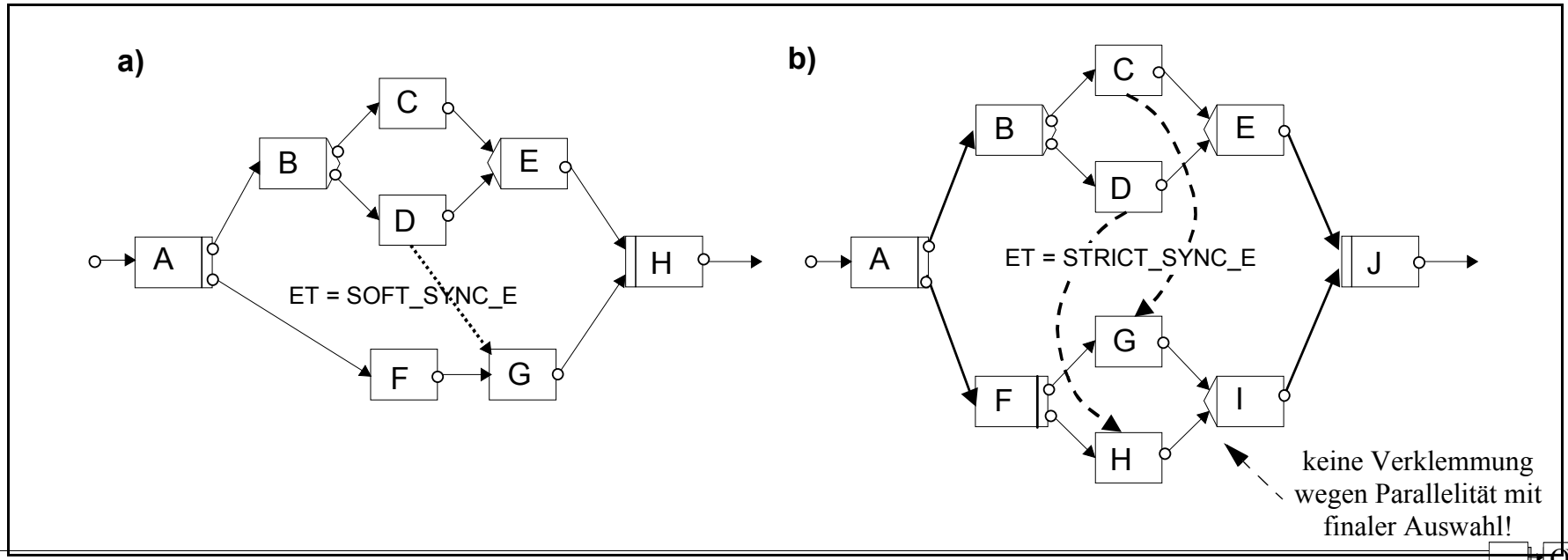
- Schleifen:



ADEPT$_{base}$: Kontrollfluß (2)

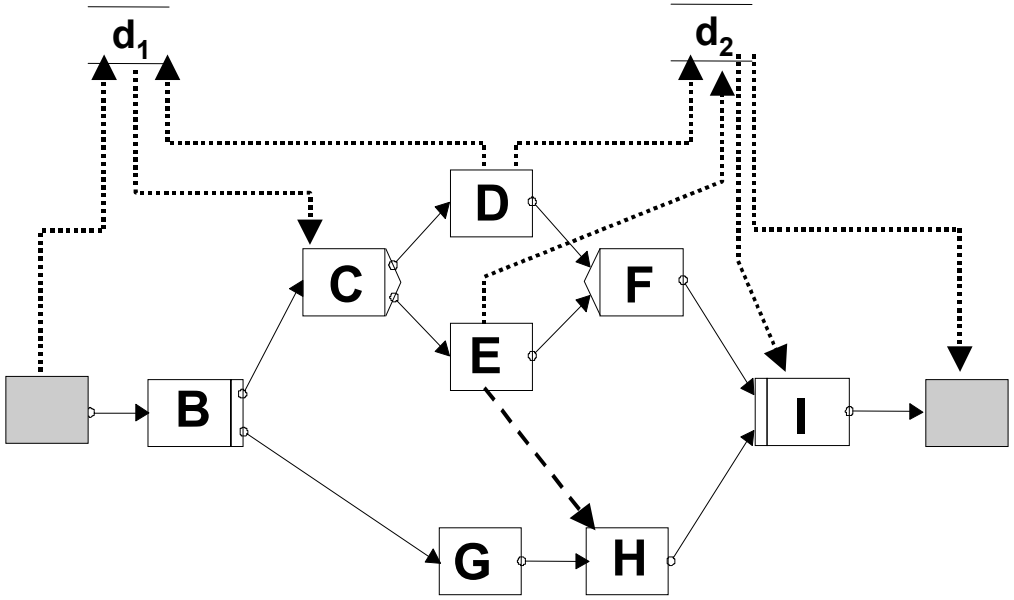
Synchronisations-Kanten

- „Einfache“ Sync-Kante X → Y (ET = SOFT_SYNC_E)
  - Beschreibt *Verzögerung* zwischen X und Y: Y darf frühestens dann aktiviert bzw. ausgeführt werden, wenn X entweder zuvor erfolgreich oder feststeht, daß X nicht mehr zur Ausführung kommt
- „Strikte“ Sync-Kante X → Y (ET = STRICT_SYNC_E)
  - *Erfolgsabhängigkeit* zwischen X und Y: Y darf nur dann aktiviert bzw. ausgeführt werden, wenn X zuvor erfolgreich beendet worden ist
- Anwendungen: Modellierung gewünschter Reihenfolgebeziehungen, Vermeidung nebenläufiger Schreibzugriffe auf Daten



ADEPT$_{base}$: Datenfluß

- Abbildung zwischen Knoten-Parametern und globalen Daten-Elementen (→ Datenfluß-Definition)
- Unterschiedliche Typen von Knoten-Parametern: *Optional* vs. *Mandatory*, *Requestable*, ...
- Wohldefinierte Regeln für Korrektheit des Datenflusses, Effizienz-Überprüfungen

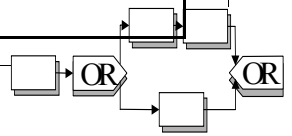


d — data element

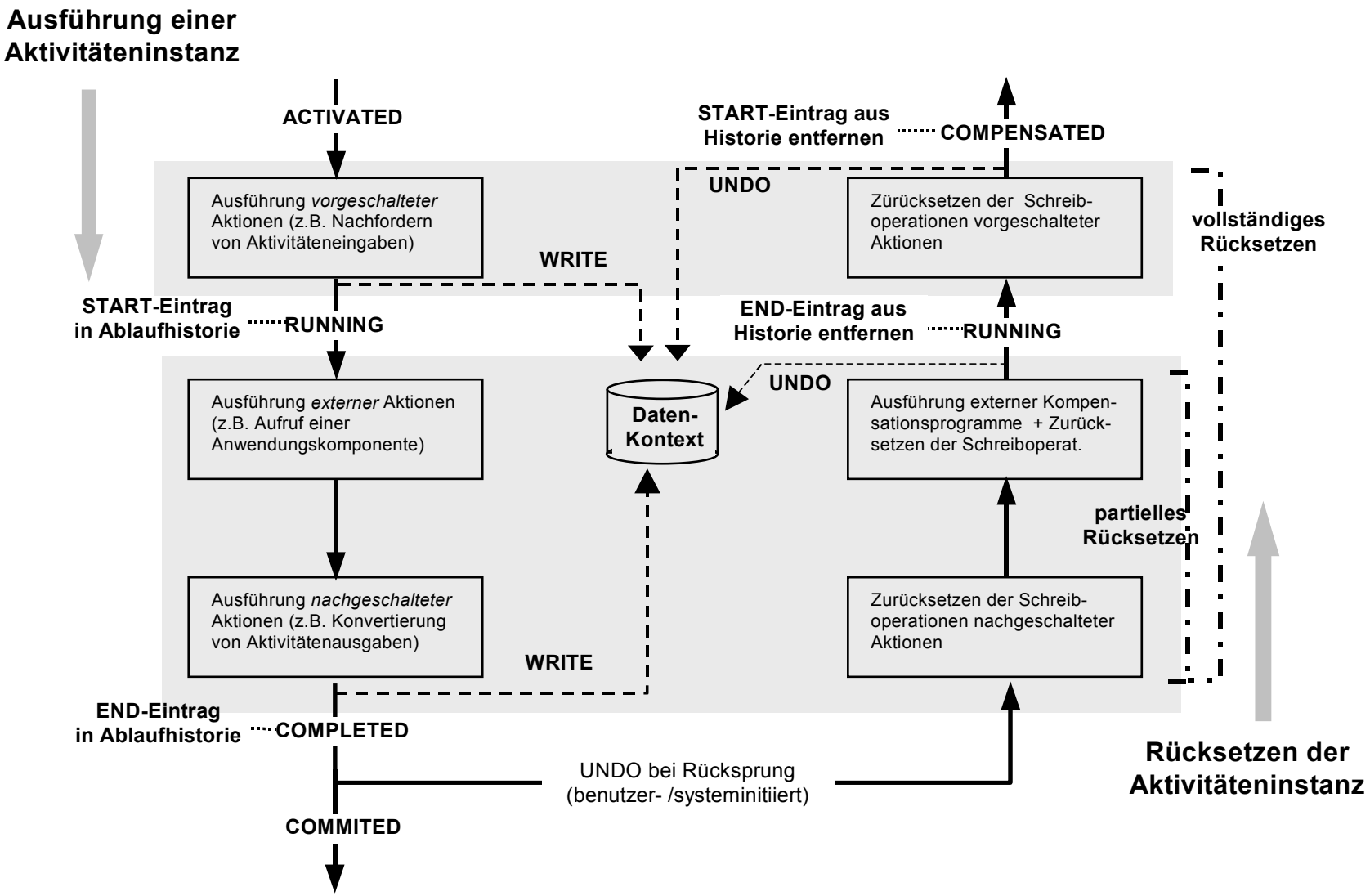
data edge

DF = { (d_1, STARTFLOW, par_1, write),
(d_1, C, par_2, read), (d_1, D, par_3, write),
(d_2, D, par_5, write), (d_2, E, par_6, write),
(d_2, I, par_7, read), (d_2, ENDFLOW, par_8, read),
}

keine expliziten Komponenten-Referenzierungen (z.B. $d_1.x$)

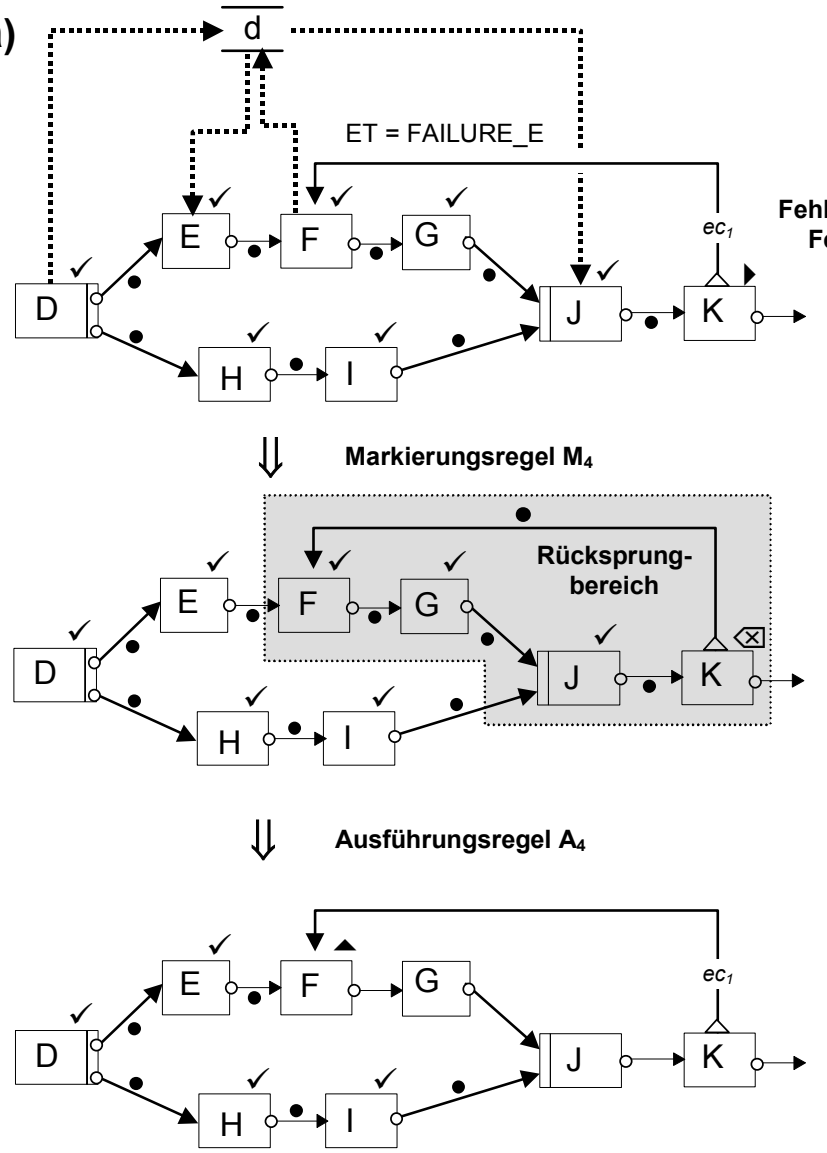


ADEPT$_{base}$: Ausführungsmodell



Vormodellierung von Ausnahmen: Rückwärtssprünge

a)



✓ NS = COMPLETED

⌫ NS = FAILED

▲ NS = ACTIVATED

▶ NS = RUNNING

● ES = TRUE_SIGNALED

b)

Historie des Datenelements d vor dem Rücksprung		
nodeId	nodeIt	value
F	1	10
D	1	13

⇓

Historie des Datenelements d nach dem Rücksprung		
nodeId	nodeIt	value
D	1	13

c) Ablaufhistorie vor dem Rücksprung (Ausschnitt):

...END(H) END(E) START(F) START(I) END(I) END(F) START(G) END(G) START(J) END(J) START(K)

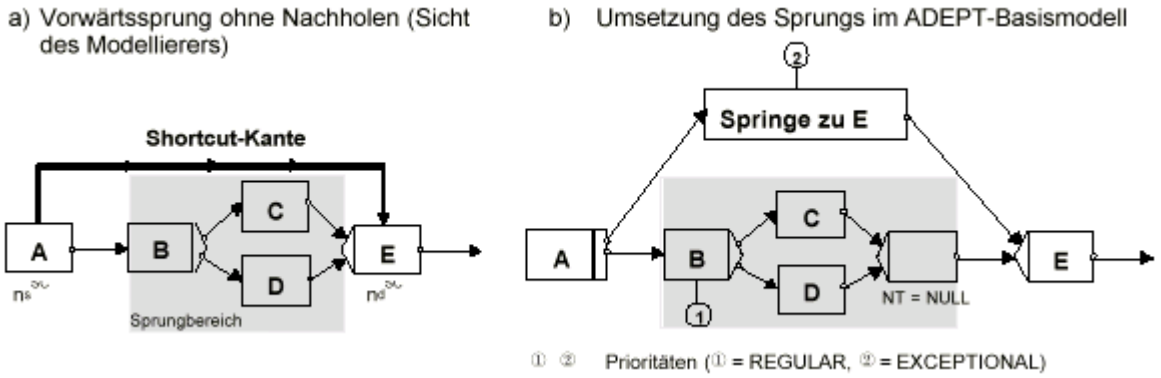
⇓

Ablaufhistorie nach dem Rücksprung (Ausschnitt):

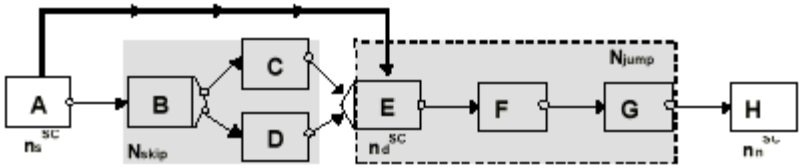
...END(H) END(E) START(I) END(I)

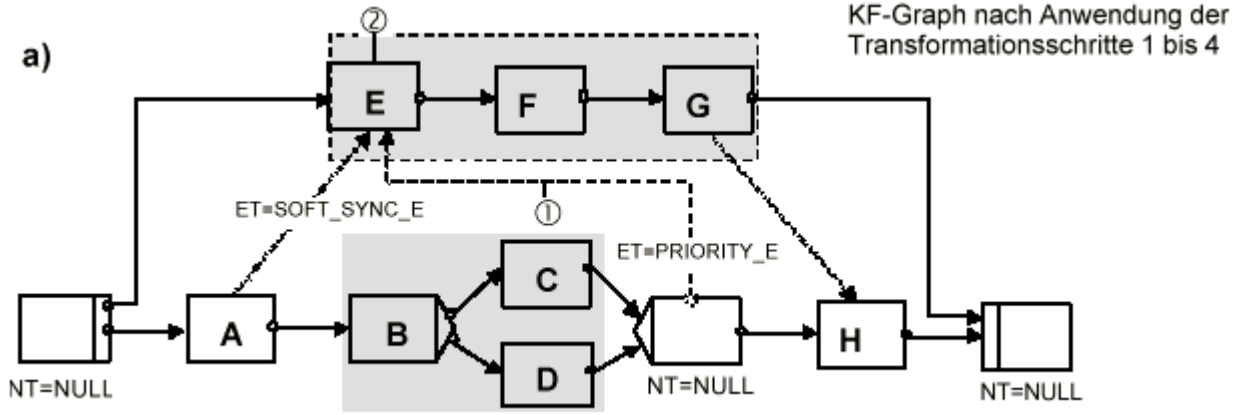
Vormodellierung von Ausnahmen: Vorwärtssprünge

- Ohne Nachholen



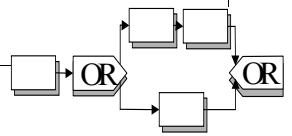
- Mit Nachholen



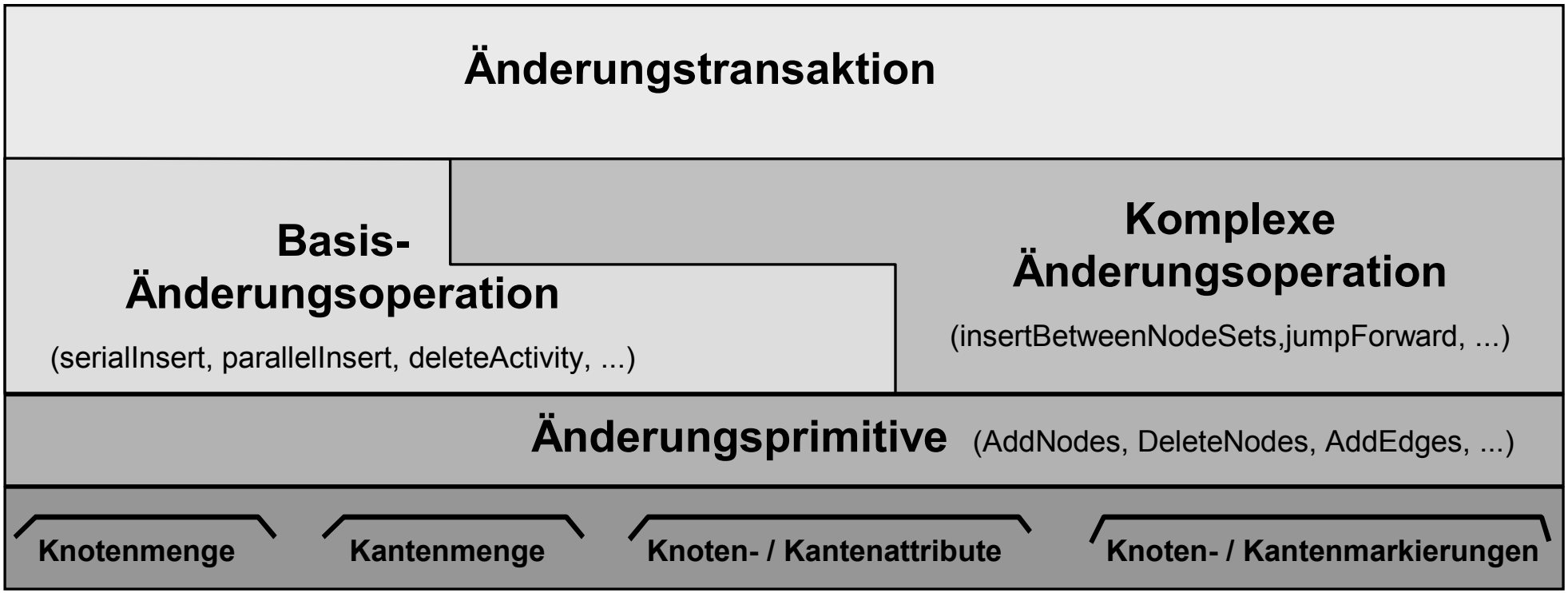


ADEPT$_{flex}$: Ad-hoc-Adaptationen

- Kernziel: Dynamische Adaptationen von Kontroll- und Datenfluß-Definitionen sowie von Workflow-Attributen und Workflow-Stati
- Vollständige und minimale Menge von *Basis*-Operatoren für
  - Hinzufügen von Knoten
  - Löschen von Knoten
  - Verschieben von Knoten
  - Dynamische Knoten-Synchronisation
  - Manipulation von Daten-Elementen und Daten-Links
- *Komplexe* Operatoren für
  - Behandlung von *Blöcken* (z.B. Einfügen einer ganzen Schleife)
  - Abstraktion von graphtechnischen Details (anwendbar für Benutzer)
- Für jeden Operator Definition von
  - formalen Vor- und Nachbedingungen
  - Graph-Transformations-Regeln mit wohl-definierter Semantik (Graph-Grammatik)
  - Mechanismen zur Erkennung mögliche Konflikte und Seiteneffekte
  - Strategien zur Behandlung solcher Konflikte und Seiteneffekte
- Ausführung jedes Operators
  - erfüllt statische und dynamische Korrektheitskriterien
  - verletzt Datenkonsistenz nicht
  - resultiert in einem zulässigem Workflow
  - ist transaktional

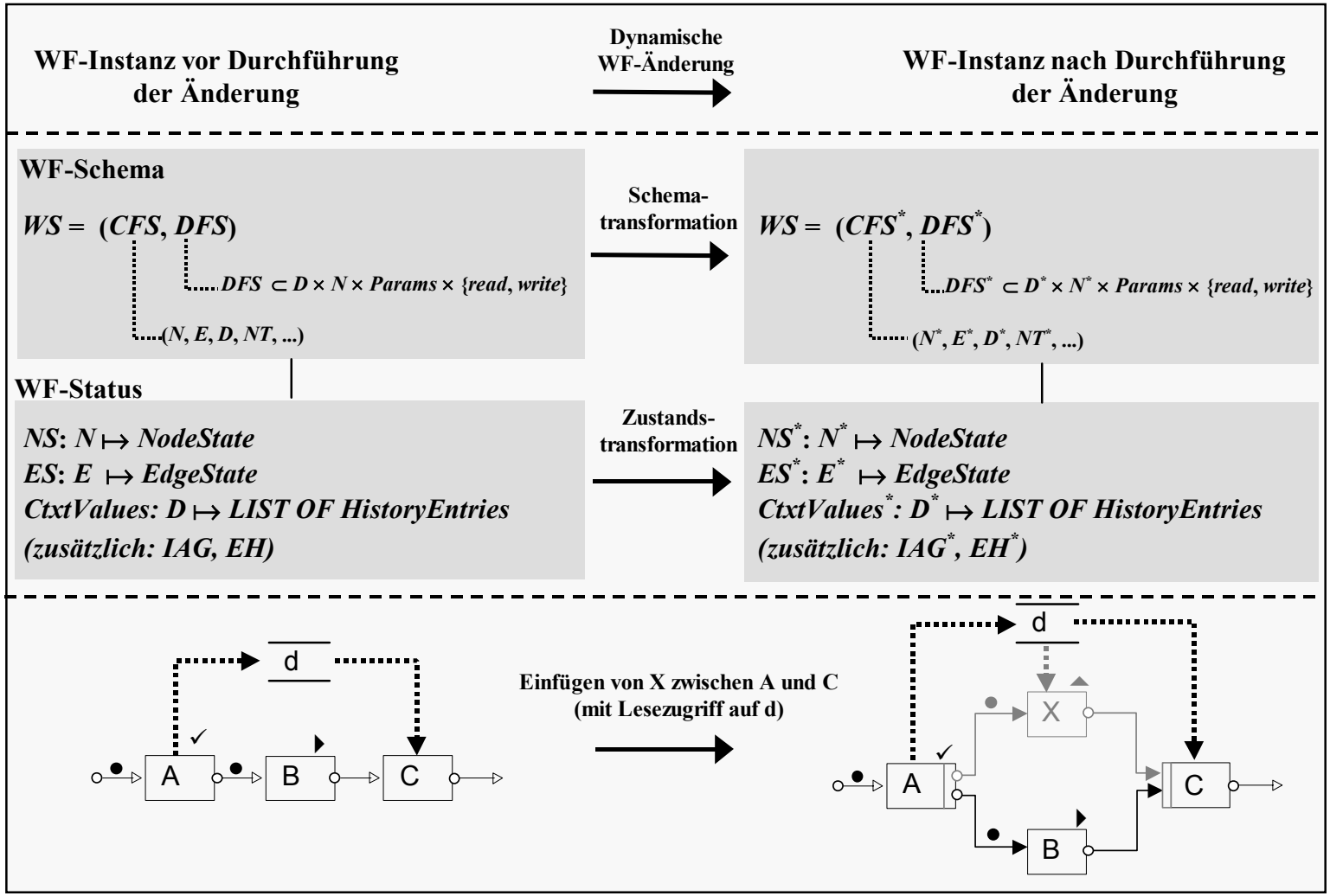


ADEPT$_{flex}$: Operator-Hierarchie



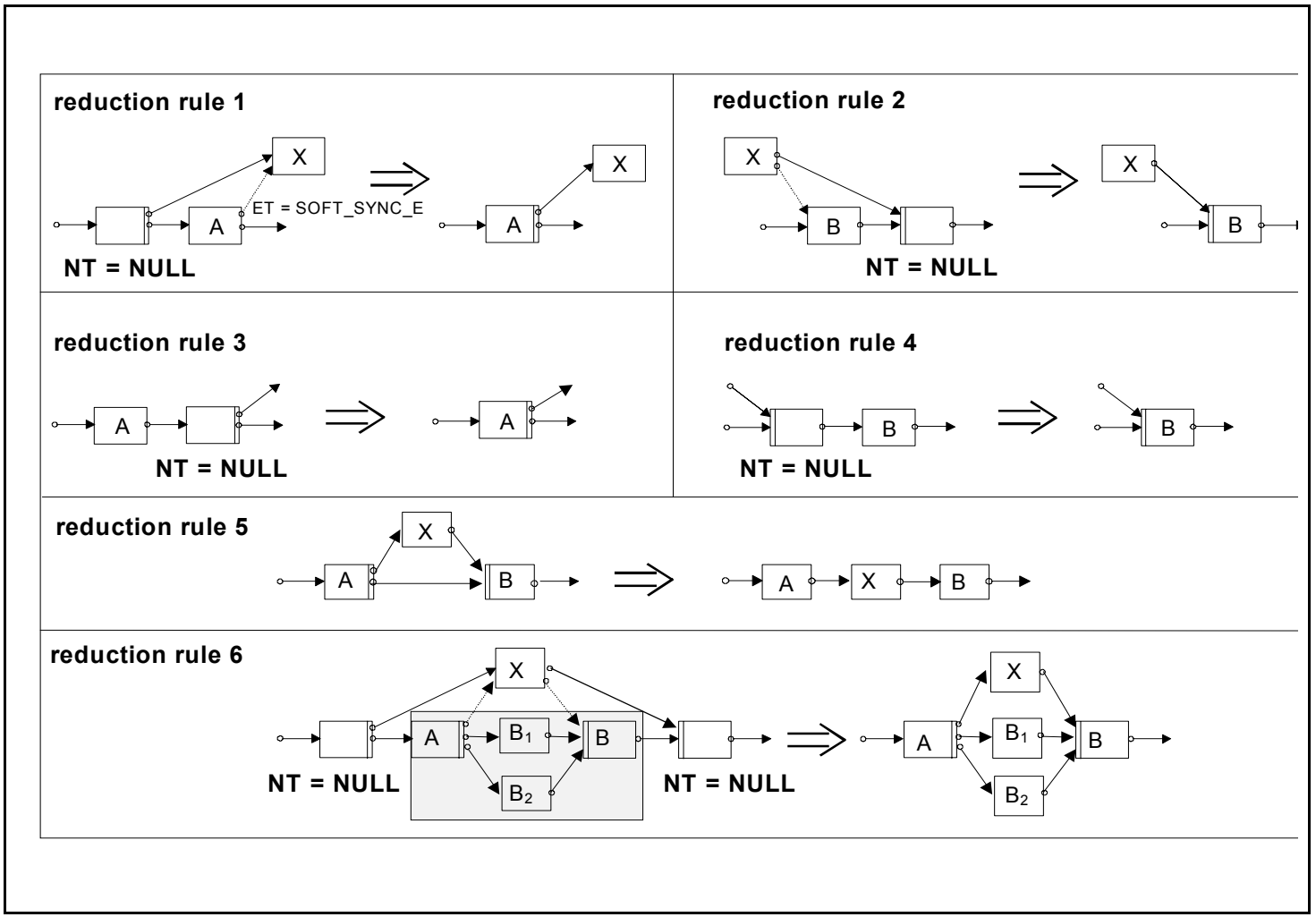
Stufen einer Adaptation

Legende:
CSF: Control flow Schema
DSF: Data flow schema
Ctxt: Context
IAG: Instanz-Abhän-gigkeitsgraph
EH: Ablaufhistorie

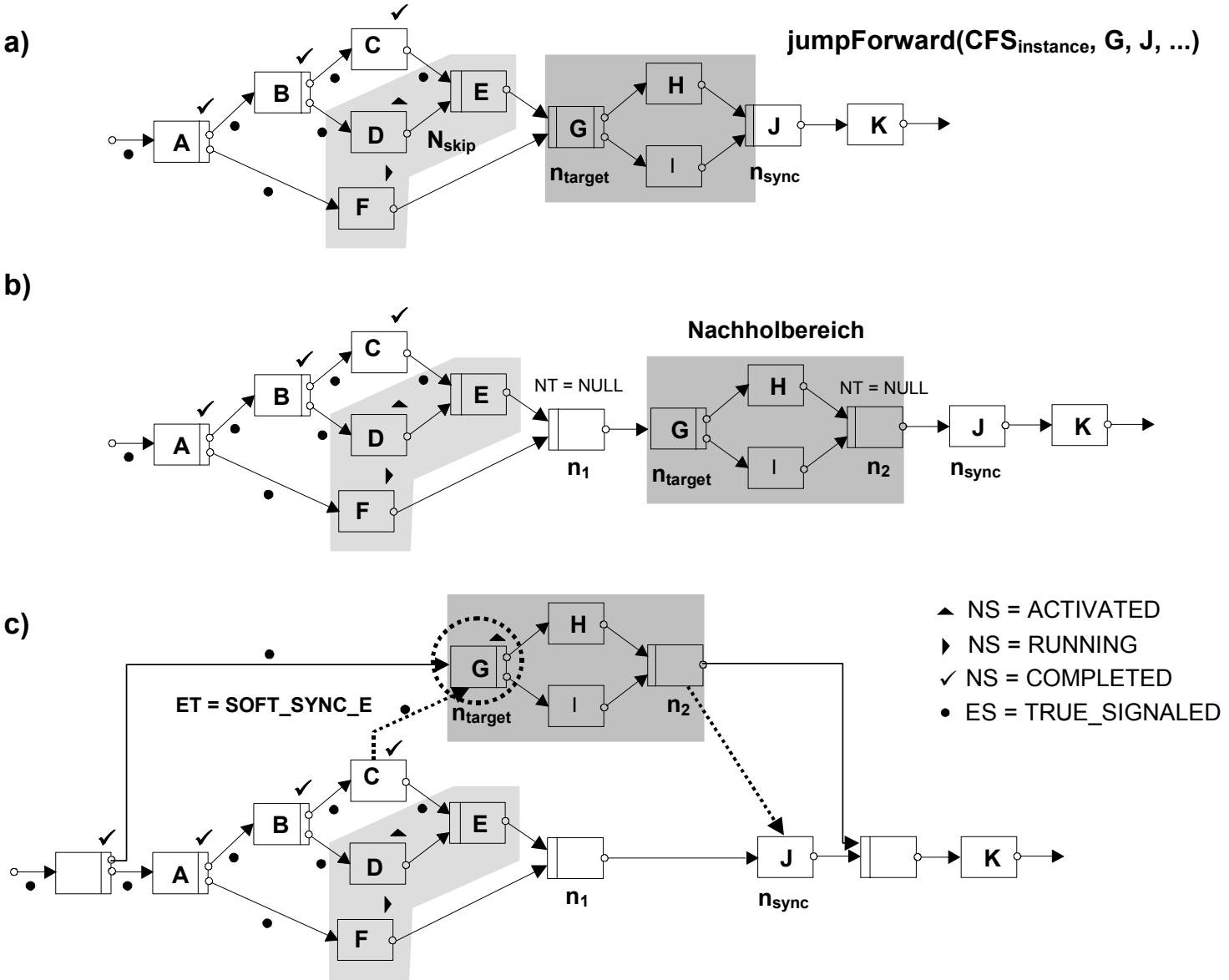


ADEPT$_{flex}$: Reduktions-Regeln

- Dienen der syntaktischen Vereinfachung von Workflow-Definitionen
- Werden nach dynamischen Adaptionen eingesetzt

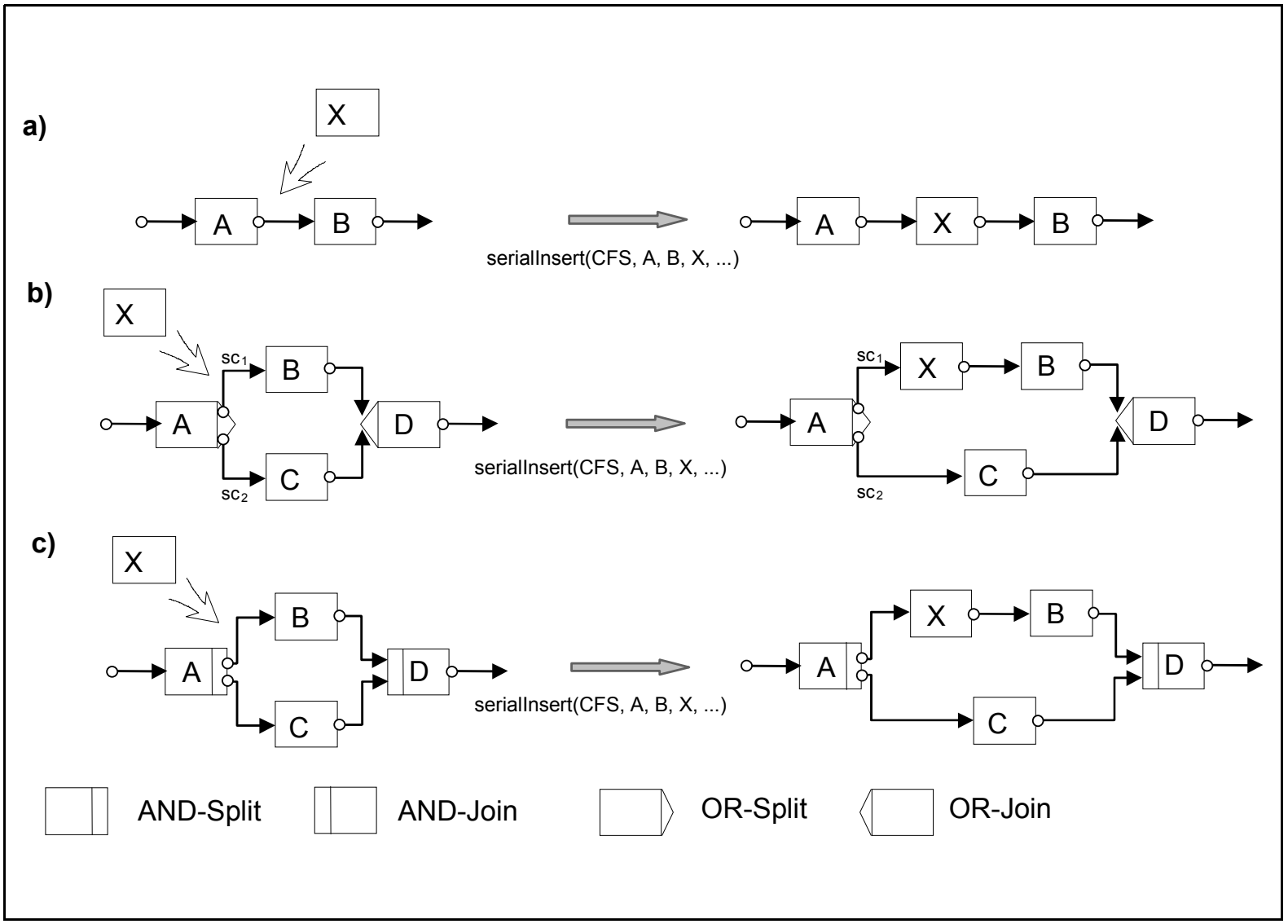


ADEPT$_{flex}$: Ad-hoc-Sprünge



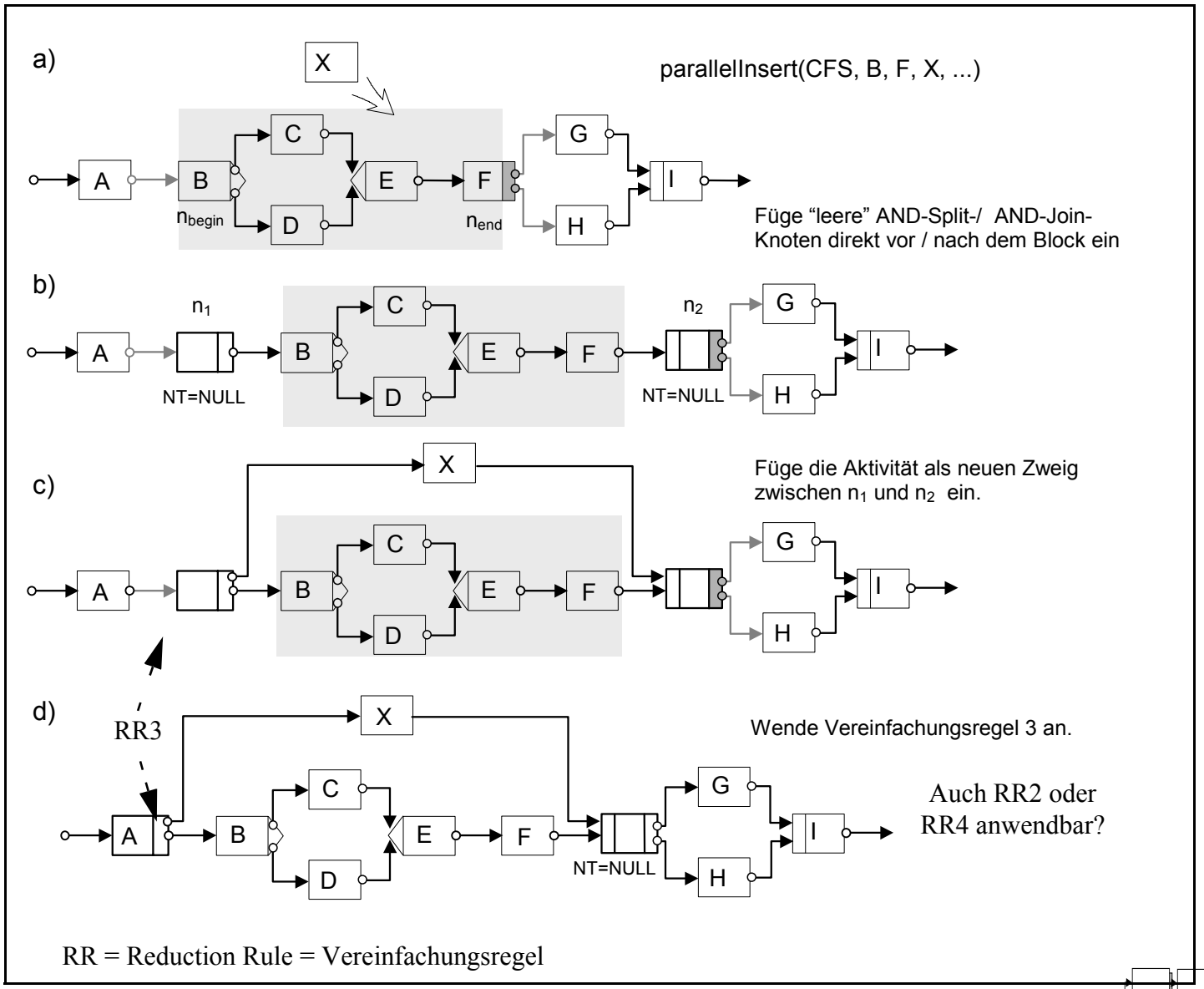
ADEPT$_{flex}$: Einfügen von Knoten (1)

- Serielles Insert:

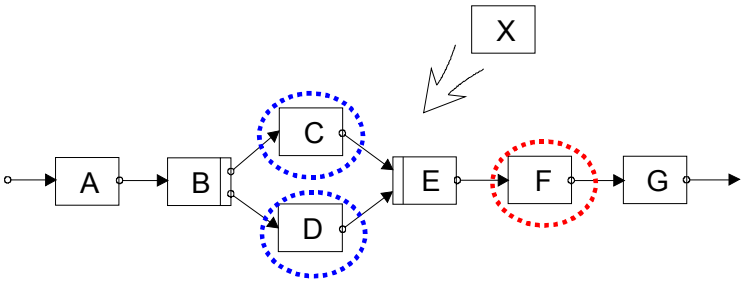


ADEPT$_{flex}$: Einfügen von Knoten (2)

- Paralleles Insert:
  - Vorteil: Weniger Ausführungsverzögerng als bei seriellem Insert
  - Nachteil: Resourcenabhängig

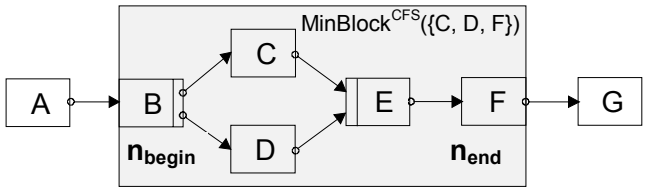


$ADEPT_{flex}$: Einfügen von Knoten zwischen zwei Knotenmengen

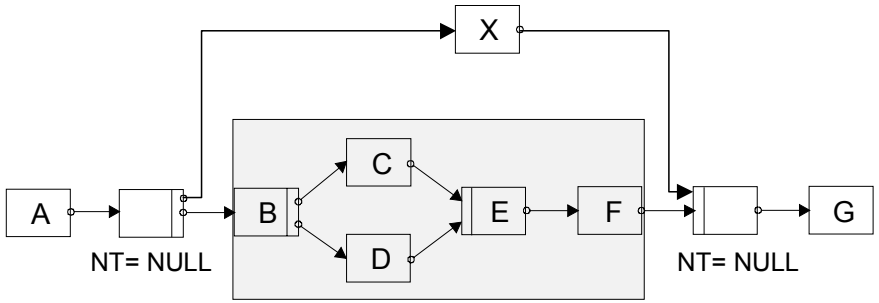


Einfügen von X zwischen
M_{before} = {C, D} und M_{after} = {F}

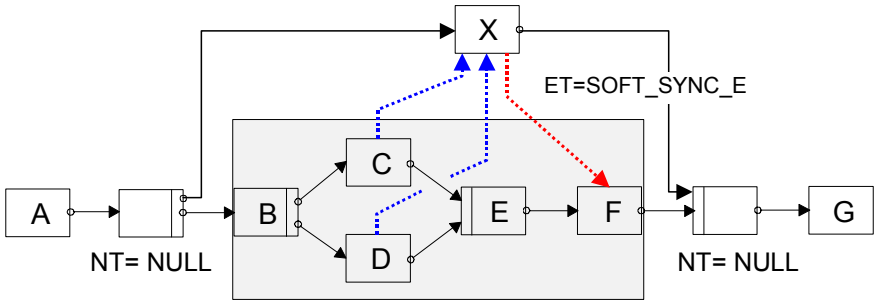
Schritt 1: Minimalen Block bestimmen



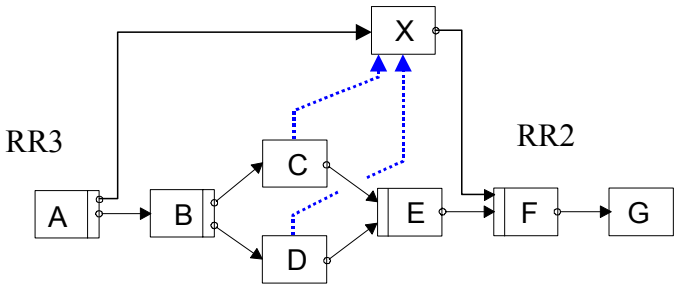
Schritt 2: X parallel zu diesem Block einfügen



Schritt 3: Sync-Kanten einfügen



Schritt 4: Vereinfachungsregeln anwenden

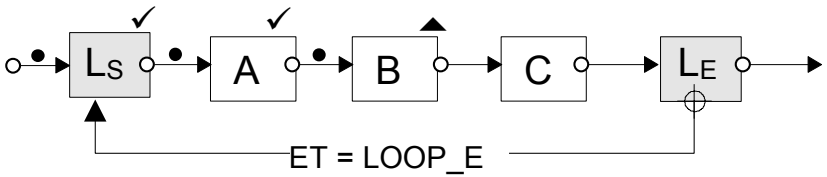


ADEPT$_{flex}$: Einfügen von Knoten - Zustandsvergabe

(Parallel zu Bereich zwischen B und B, also parallel zu B)

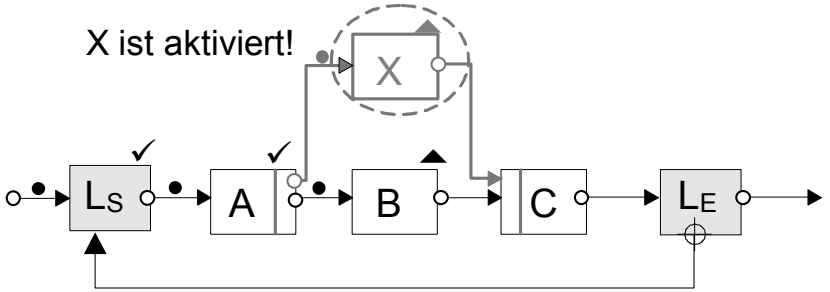
a)

Ausführungsgraph CFS_{inst}

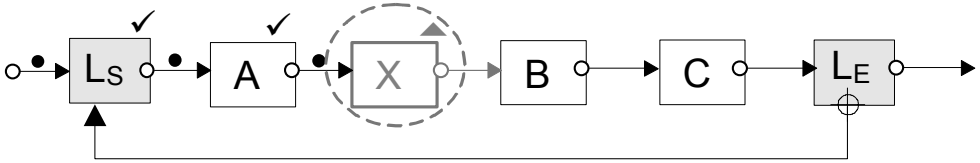


b) parallelInsert(CFS_{inst}, B, B, X, ...)

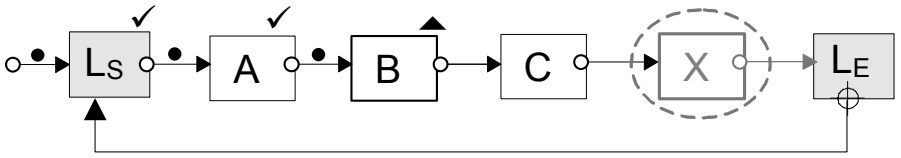
X ist aktiviert!



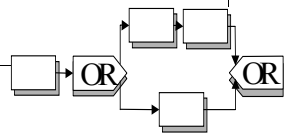
c) serialInsert(CFS_{inst}, A, B, X, ...)



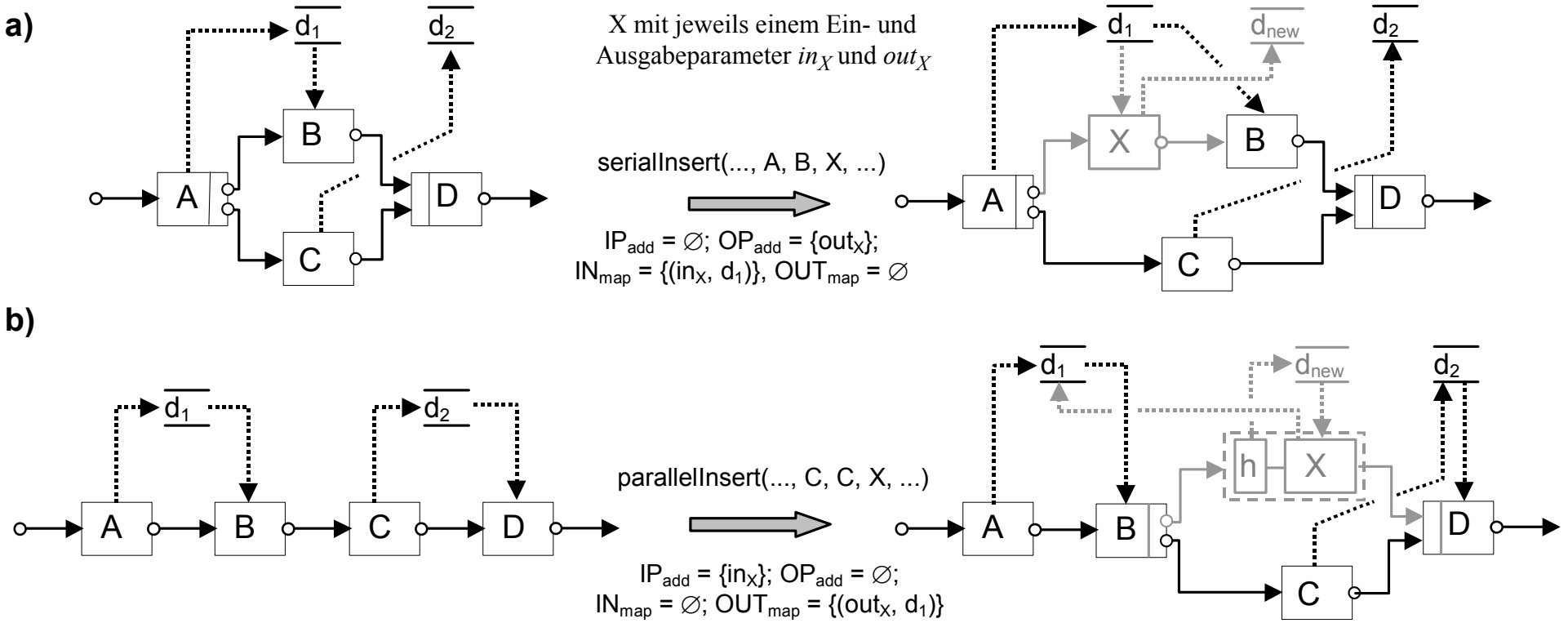
d) serialInsert(CFS_{inst}, C, L_E, X, ...)



▲ NS = ACTIVATED ✓ NS = COMPLETED ● ES = TRUE_SIGNALED

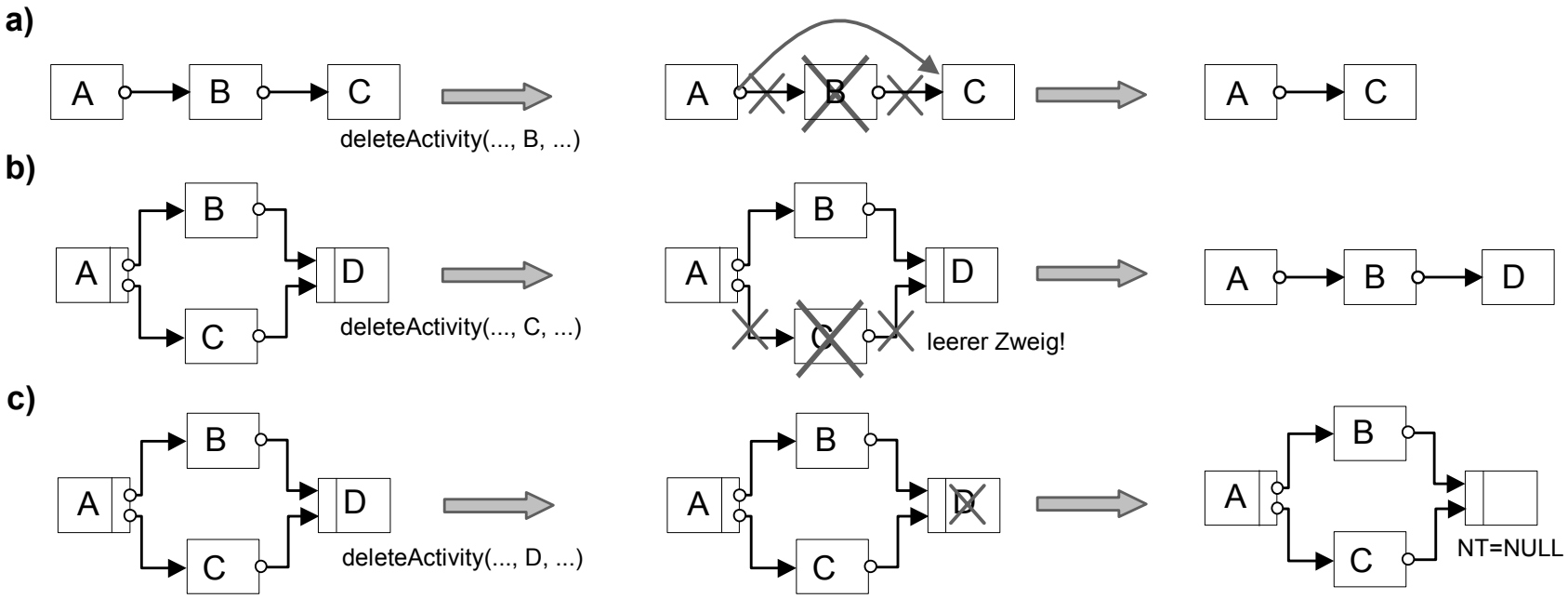


ADEPT$_{flex}$: Einfügen von Knoten - Anpassung Datenfluß

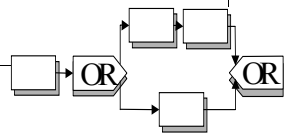


- IP_{add} : Eingabeparameter von X, deren Versorgung durch vorgeschalteten, interaktiven Nachforderungsdienst erfolgen soll
- OP_{add} : Ausgabeparameter von X, die an neu zu generierendes Datenelement geknüpft werden sollen
- $IN_{map} \subset InParams \times D$: Menge von Bindungen zwischen Eingabeparametern und globalen Datenelementen; $(par, d) \in IN_{map}$ $\Rightarrow$ es soll Lesekante von *par* zum Datenelement *d* eingefügt werden
- $OUT_{map} \subset OutParams \times D$: Menge von Bindungen zwischen Ausgabeparametern und globalen Datenelementen; $(par, d) \in OUTmap \Rightarrow$ es soll Schreibkante von *par* zum Datenelement *d* eingefügt werden.

ADEPT$_{flex}$: Löschen von Knoten



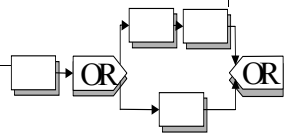
- Löschen eines leeren Zweigs bei konditionaler Verzweigung?



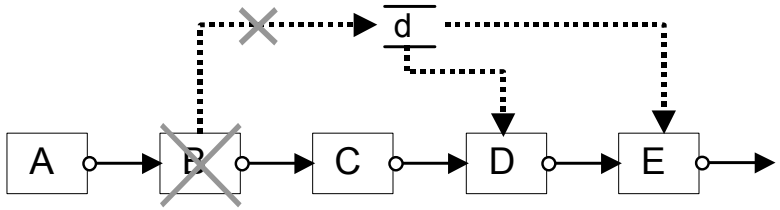
Löschen von Knoten - Anpassung Datenfluß (1)

- Problem: Durch das Löschen einzelner Aktivitäten (und ihrer Schreibzugriffe auf den Workflow-Datenkontext) können Nachfolge-Aktivitäten obligate Eingabedaten fehlen

- Strategien zur Behandlung unversorgter Parameter (nach Löschen eines Schrittes X):
  - Von *X* datenabhängige Aktivitäten werden ebenfalls aus WF-Schema entfernt (evtl. kaskadierend)
  - Zwischen *X* (bzw. seinen direkten Vorgängerknoten) und datenabhängigen Schritten werden Ersatzschritte zur Datenversorgung eingefügt
  - Aktivitäten des Workflow-Schemas werden Nachforderungsdienste vor-/nachgeschaltet, mit denen fehlende Daten interaktiv versorgt werden können
  - Anwender ändert das Datenfluß-Schema direkt mit den dafür bereitgestellten Basisoperationen ab, um wieder zu einem korrekten Datenfluß zu gelangen
  - Kombinierte Anwendung der verschiedenen Strategien



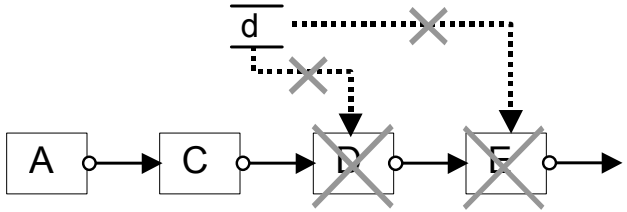
Löschen von Knoten - Anpassung Datenfluß (2)



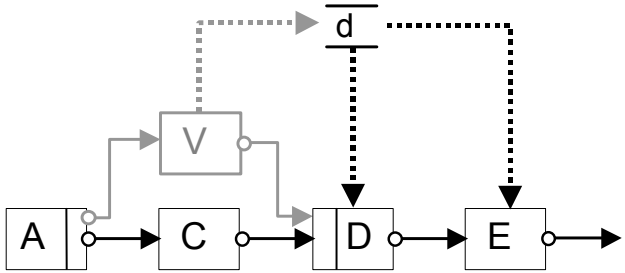
deleteActivity(*CFS*, *DFS*, *B*, ...)

Mögliche Anpassungen des DF-Schemas?

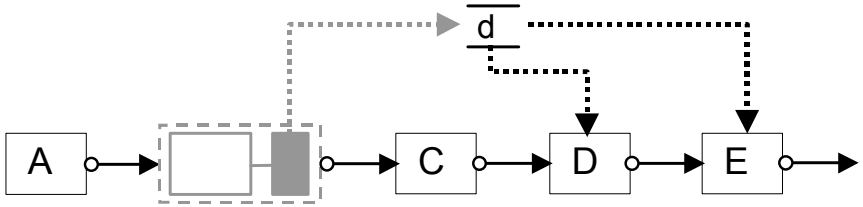
a)



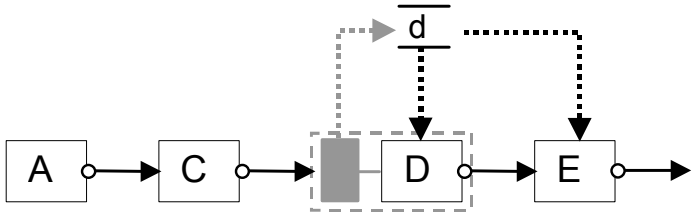
b)



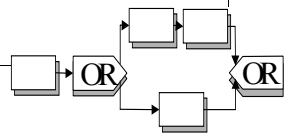
c)



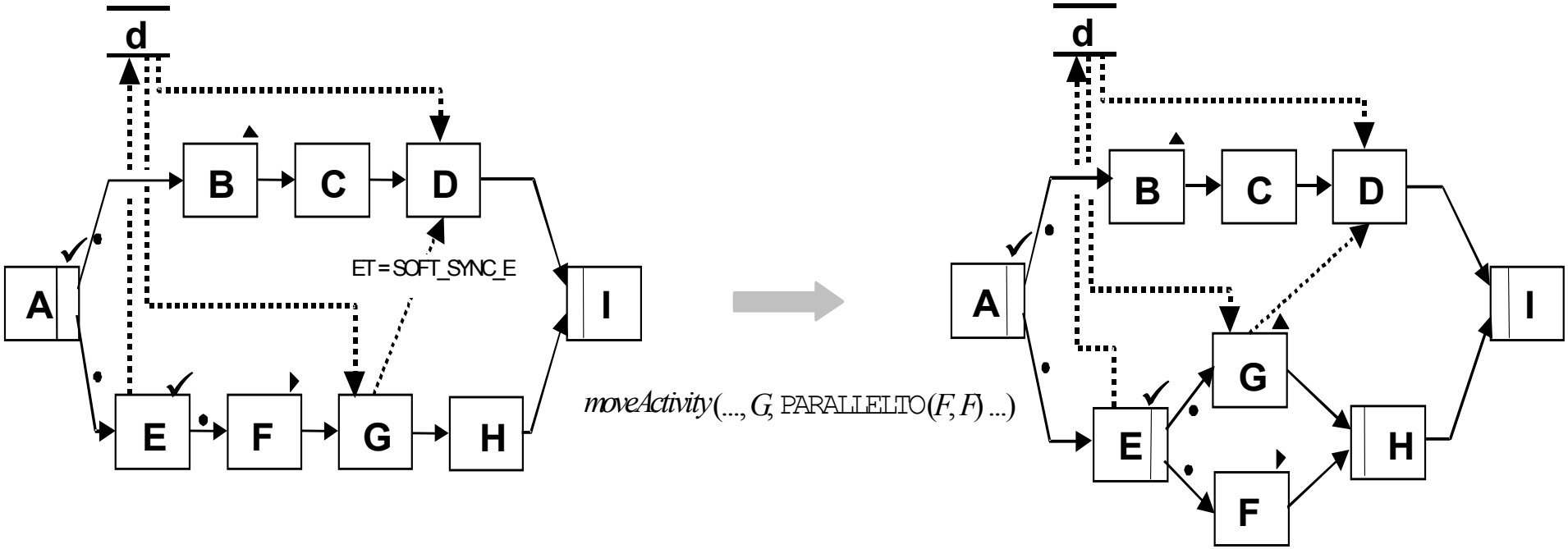
d)



a) Löschen datenabhängiger Aktivitäten
b) Einfügen einer Versorgeraktivität *V*
c) Sofortiges Nachfordern der fehlenden Daten durch Aktivierung eines nachgeschalteten Dienstes
d) Nachfordern der Daten durch Aktivierung eines vorgeschalteten Nachforderungsdienstes

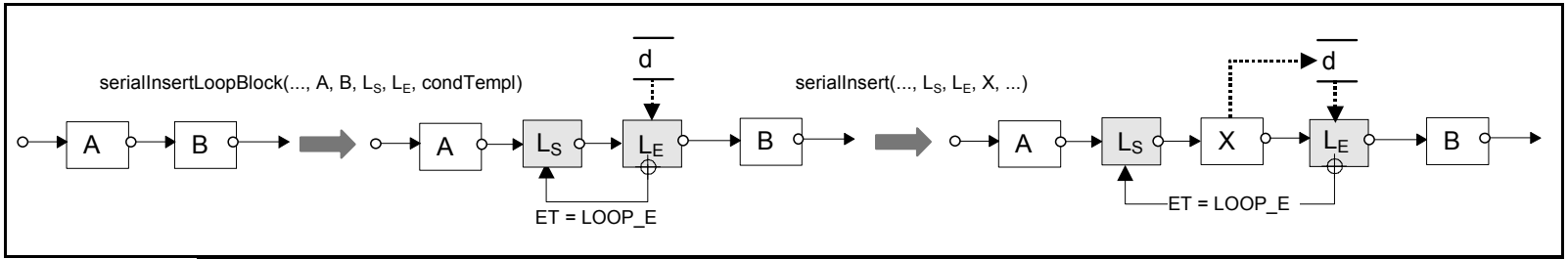


ADEPT$_{flex}$: Verschieben von Knoten

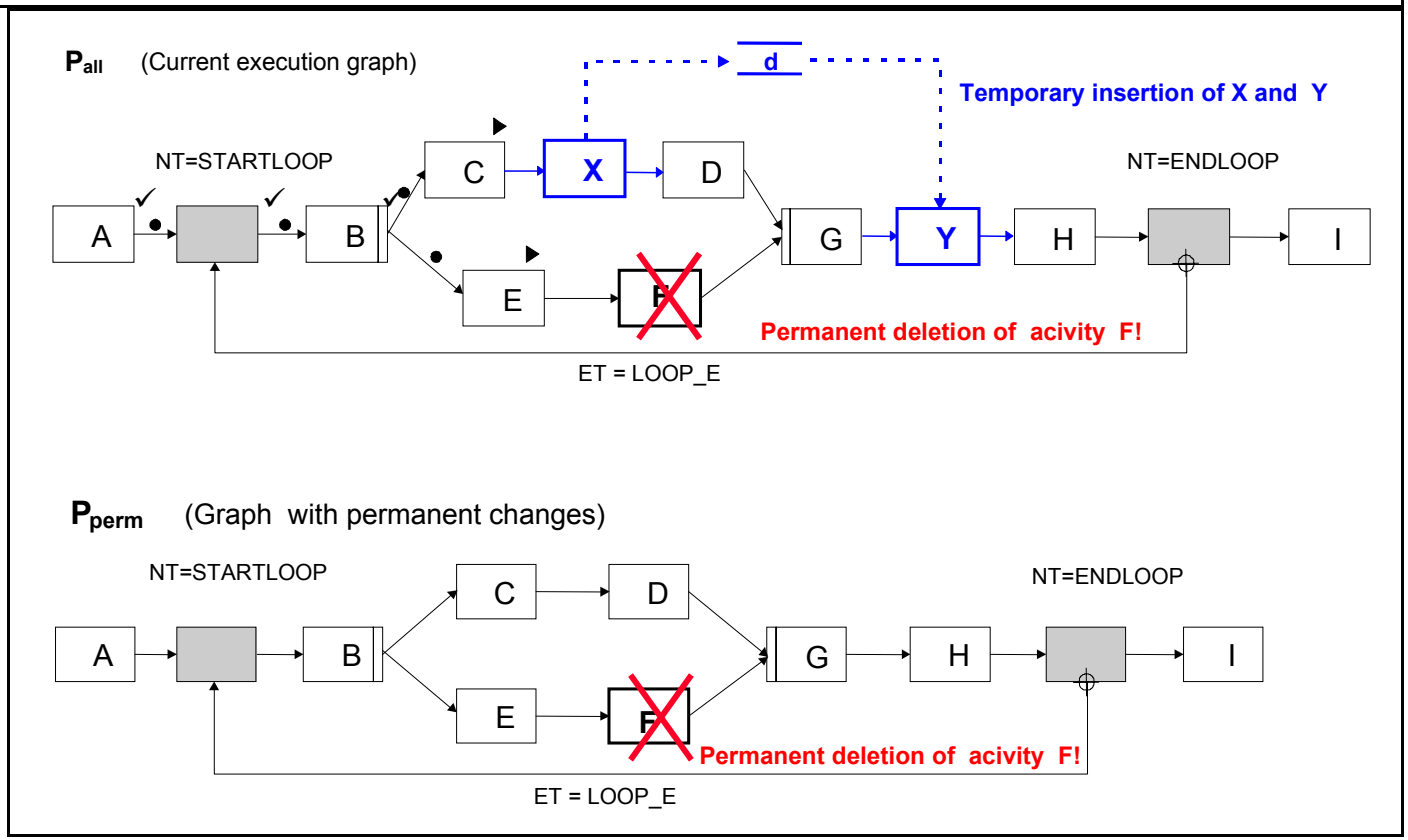


ADEPT$_{flex}$: Adaptation von Schleifen

- Generierung von Schleifen

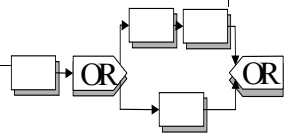


- Temporäre versus permanente Schleifen-Adaptation:
  - *Temporär*: Adaptation gilt nur für aktuellen Schleifen-Durchlauf
  - *Permanent*: Adaptation gilt für alle weiteren Durchläufe
- Temporär/Permanent-Problematik relevant auch außerhalb von Schleifen bei partiellen Rollbacks



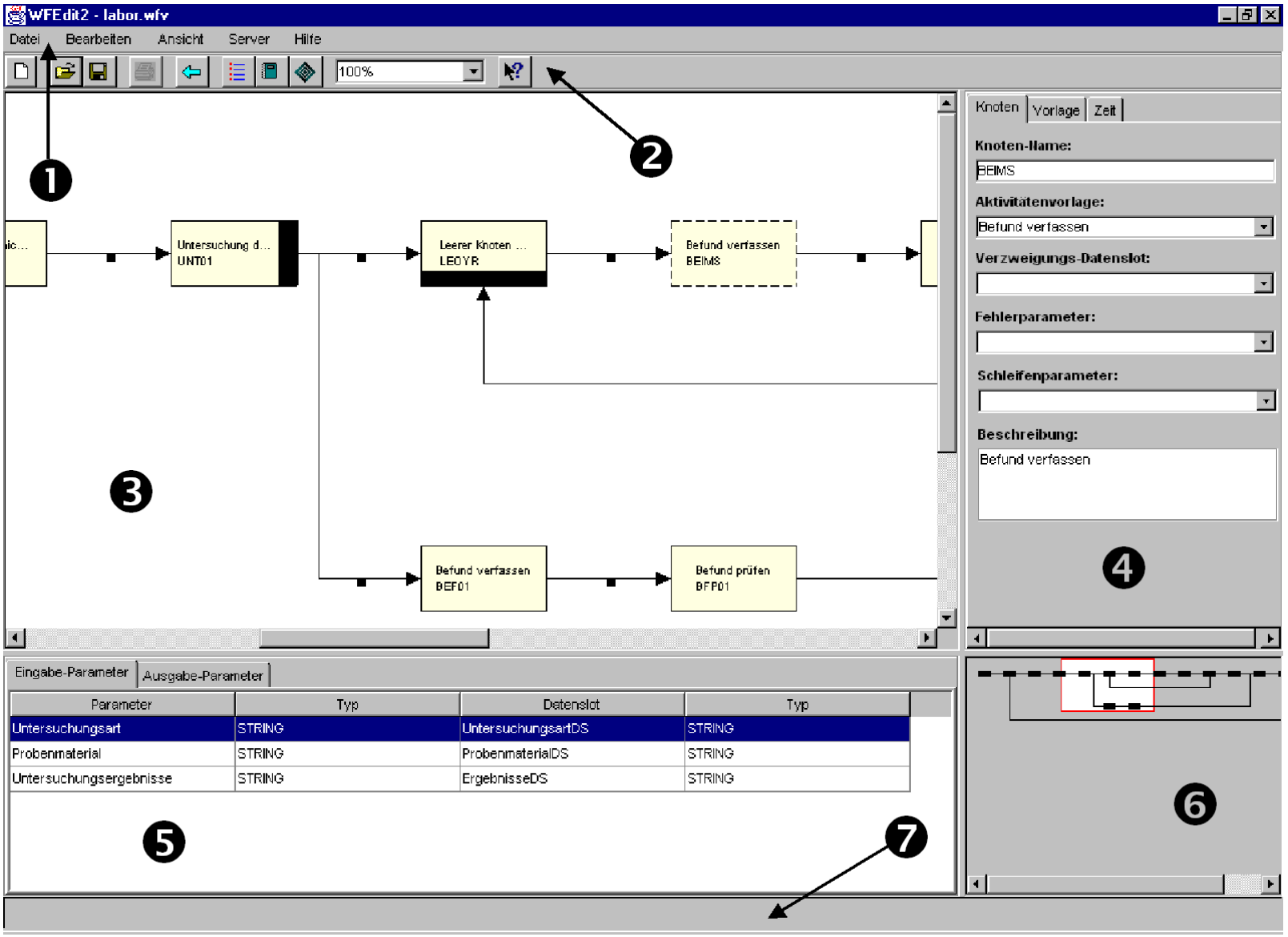
ADEPT: Implementierung

- ADEPT-WF-Modeler
  - Syntax-getriebener, graphischer Editor für Workflow-Definitionen
  - Validierung von Korrektheits-Kriterien
  - Konstrukte für Vor-Modellierung von vorhersehbaren Ausnahmen (z.B. durch backward / forward Sprünge im Kontrollfluß)
- ADEPT-WF-Server:
  - Unterstützung von dynamischen Adaptationen einzelner Workflow-Instanzen
  - Umfassendes API für Implementierung von Workflow Clients
- Verteilte Version: $\text{ADEPT}_{\text{distributed}}$
- Implementierung aller Komponenten auf der Basis von JAVA und RDBMS

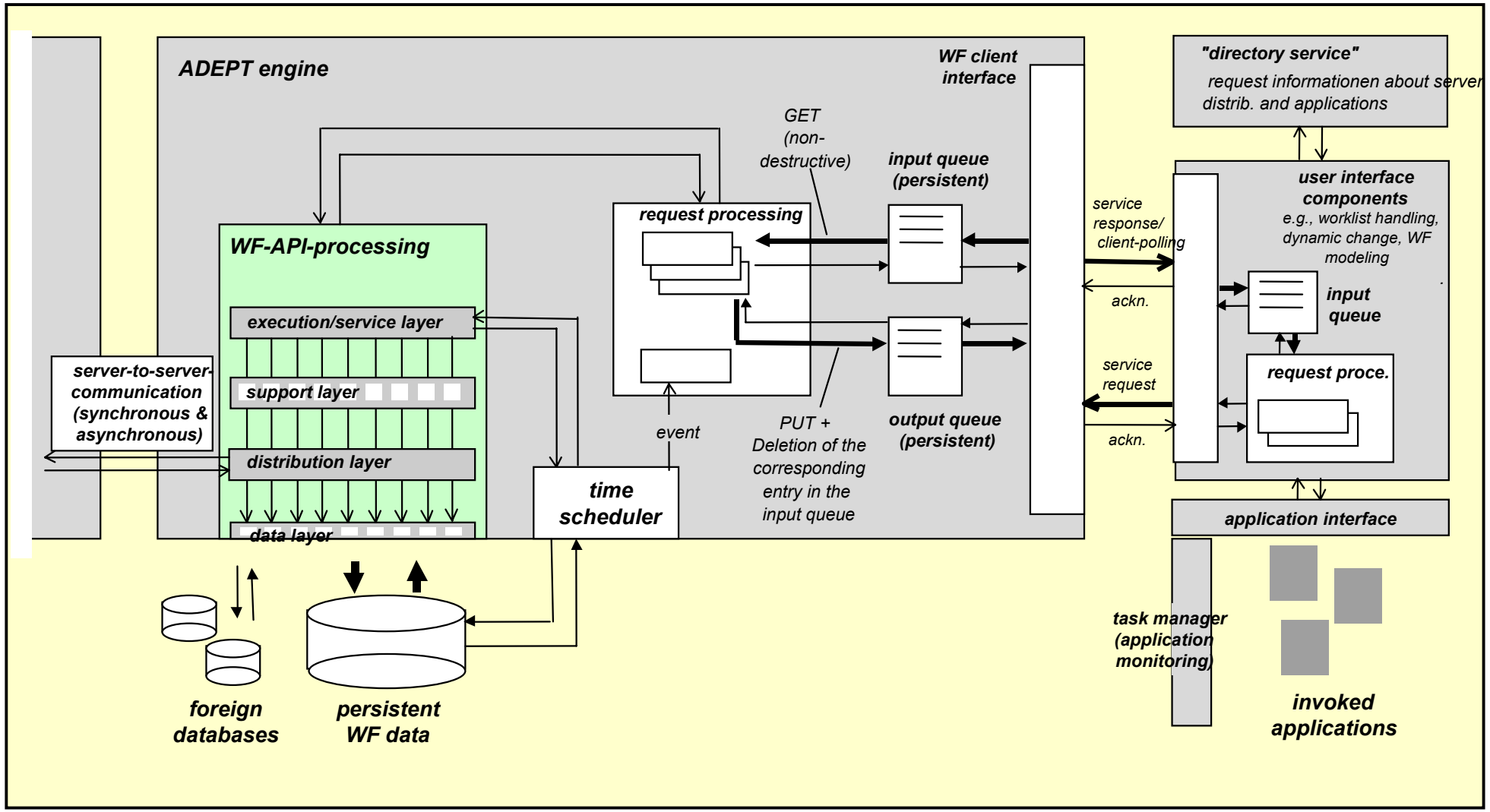


ADEPT: Workflow-Editor

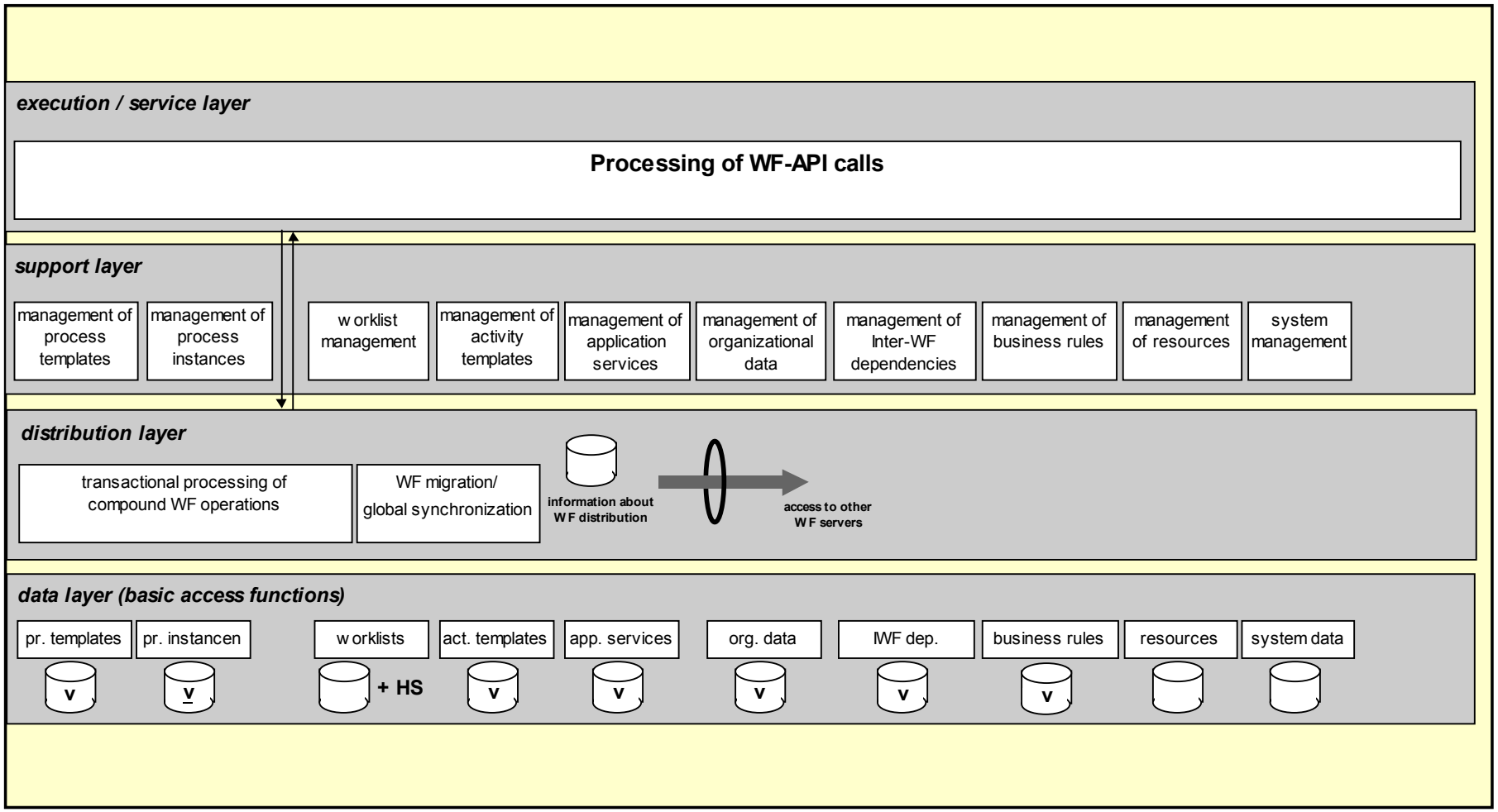
- Hauptmenü (1)
- Symbolleiste (2)
- Aktuell bearbeitete Workflow-Vorlage (3)
- Detailinformationen zu Knoten und Kanten (z.B. Bearbeiter, Zeitattribute) (4)
- Datenflusses in tabellarisch Darstellung (5)
- Workflow-Übersicht (6)
- Statusleiste (7)



ADEPT: Architektur (vereinfacht)

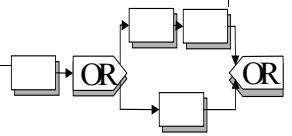


ADEPT: Architektur-Schichten



ADEPT: Zusammenfassung und Diskussion

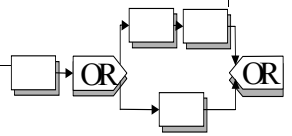
- Mächtiges formales Modell zur dynamischen Adaptation von Workflow-Instanzen
- Dreistufige Adaptation:
  1. Adaptation des Kontrollflusses
  - Vergabe von Knoten- und Kanten-Zuständen
  - Anpassung des Datenflusses
- Umfangreiches Operator- und Regelwerk zur *korrekten* und *konsistenten* Adaptation
- Ansatz orthogonal zur Frage, *welche* Adaptationen geeignet sind
- Keine Tendenz zur automatischen Adaptation
- Keine ereignisorientierte Adaptation
- Keine Behandlung der Inter-Workflow-Implikationen von Adaptationen



CHIMERA-EXC

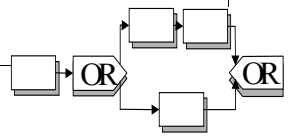
(Casati & Ceri, HP & Universität Mailand)

- Ereignisorientiere Ad-Hoc-Behandlung von logischen Ausnahmesituationen
- Spezifikation von ECA-Regeln in CHIMERA-EXC
  - Objekt-orientierte Erweitung von DATALOG
- Ausnahme-orientierte Meta-Steuerung von Workflows durch ECA-Regeln
- Implementiert auf Basis des kommerziellen Workflow-Systems FORO
- EU-Projekt WIDE
  - Später in diesem Kapitel: Workflow-Evolution in WIDE

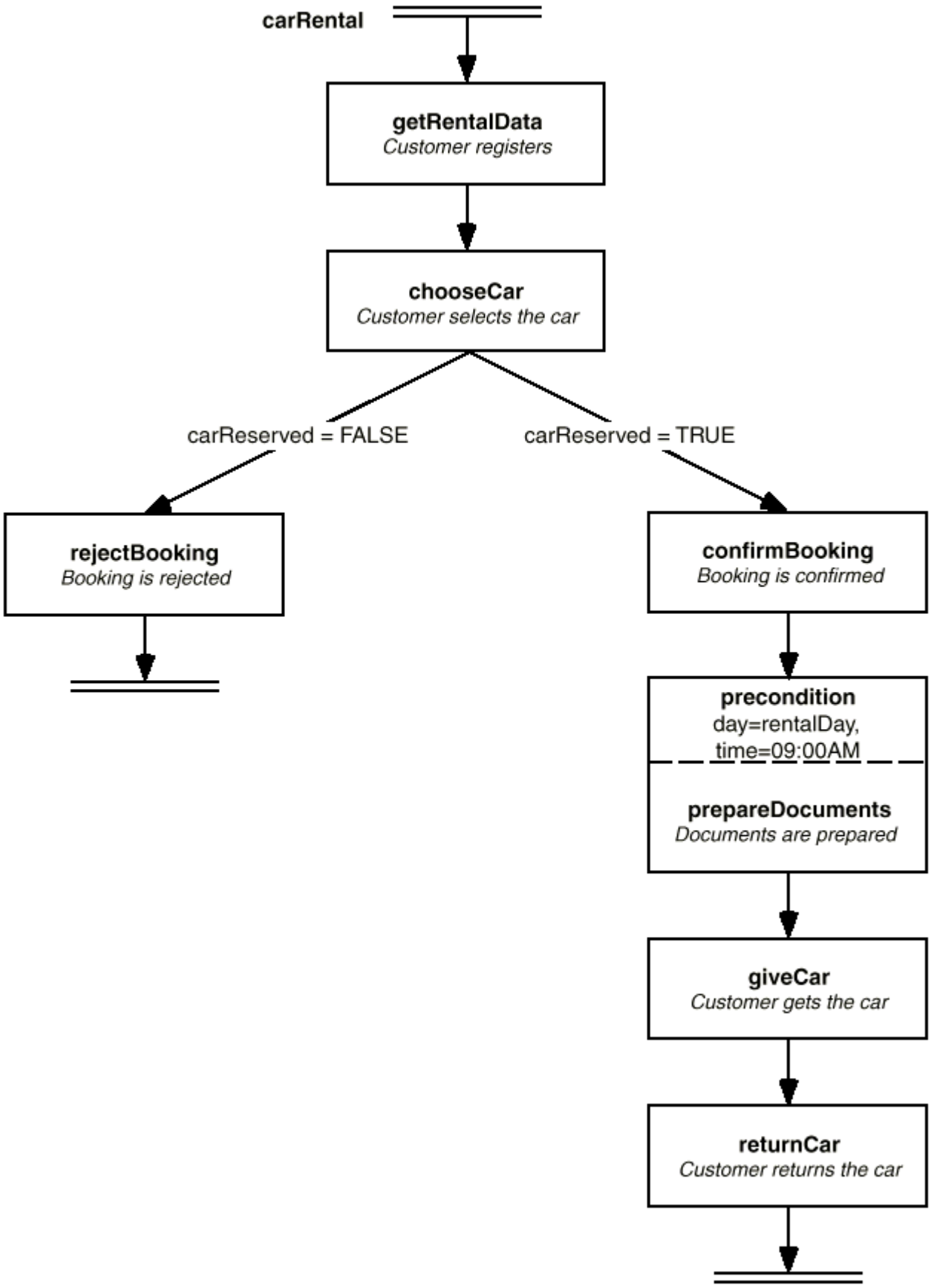


CHIMERA-EXC: Ereignisklassifikation

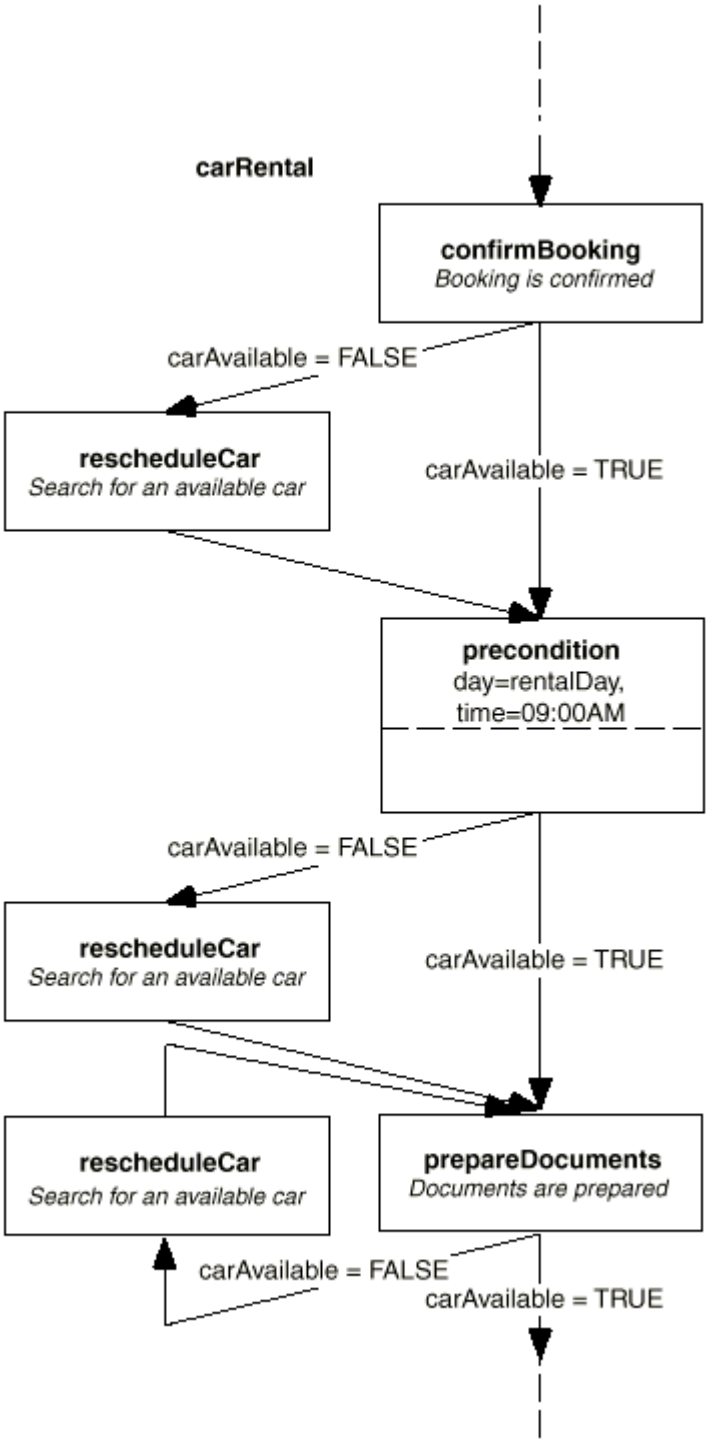
- Datenmanipulations-Ereignisse
  - Table insert/update/delete
  - Beispiel: *modify(carRental.returnTime)*
- Externe Ereignisse
  - Ausgelöst durch externe Anwendungsprogramme
  - Ereignis-Registrierung
  - Beispiel: *raise(carAccident)*
- Workflow-Ereignisse
  - Start, Beendigung oder Abbruch von Workflows oder Aktivitäten
- Temporale Ereignisse
  - Zeit-Instanz-Ereignis, z.B. Eintritt eines konkreten Datums
  - Periodisch, z.B. *date '12/25/1997' < 1/days during weeks < date '12/25/1998'* = "Jeder Sonntag zw. Weihnachten 1997 und 1999"
  - Interval-orientiert, z.B. *elapsed(interval 60 days) since E* = Ablauf von 60-Tagefrist seit *E*



Beispiel-Workflow



Vormodellierung von Ausnahmen

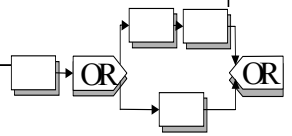


CHIMERA-EXC: Beispiel-Regeln

```
define trigger lateCarReturn
        events      modify(carRental.returnTime)
        condition   carRental(C1), occurred (modify(carRental.returnTime), C1),
                    carRental(C2), C1.bookedCarPlate = C2.bookedCarPlate,
                    C1.returnTime > C2.rentalTime
        actions     notify (C2.responsible, "Need of rescheduling car " + C2.bookedCarPlate)

define trigger customerCancel
        events      raise("customerCancel")
        condition   carRental(C), externalEvent(E), occurred(raise ("customerCancel"), E),
                    C.reservationNumber = E.parameter1
        actions     notify (C.responsible, "Customer cancelled reservation: " = E.parameter1),
                    startTask(C, rejectBooking)

define trigger carAccident
        events realtime raise("carAccident")
        condition   externalEvent(E), carRental(C1), occurred(raise("carAccident"), E),
                    E.parameter1 = C1.reservationNumber, carRental(C2), C1.bookedCarPlate = C2.bookedCarPlate
        actions     notify (C1.responsible, "Accident for car " + C1.bookedCarPlate),
                    notify (C2.responsible, "Need of rescheduling car " + C2.bookedCarPlate),
                    startCase(Accident, C1.bookedCarPlate) // Starte Workflow namens Accident
```



CHIMERA-EXC: Bewertung

- Ereignis-Workflow-Kopplung für Ausnahmebehandlung
- Unklar, wie strukturelle Adaptation durchgeführt werden
  - Beispiel: Wie wird *startTask(C, rejectBooking)* (s.o.) umgesetzt?
- Keine Zusicherung von Regelkonsistenzen
- Keine Berücksichtigung von Datenfluß-Adaptationen