ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Ausführung von variantenbehafteten Workflow-Modellen in Abhängigkeit vom Prozesskontext

Diplomarbeit an der Universität Ulm

Vorgelegt von:
Christoph Mauroner
christoph.mauroner@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert
PD Dr. Stefanie Rinderle-Ma

Betreuer:
Dipl.-Inf. Alena Hallerbach

2009

Fassung 20. Oktober 2009

Kurzfassung

Die Modellierung und Ausführung von Geschäftsprozessen sind zentrale Aufgaben in prozessorientierten Informationssystemen. In der Praxis tritt ein Prozess oftmals in zahlreichen Varianten auf, die Anpassungen an bestimmte Rahmenbedingungen darstellen. Heutige Prozessmodellierungswerkzeuge unterstützen das Modellieren und Ausführen von solchen Prozessvarianten nicht angemessen. Im Rahmen des Forschungsprojekts *Provop* (Prozessvarianten durch Optionen) bei der Daimler AG wird daher ein Lösungsansatz entwickelt, der ein transparentes und kontextabhängiges Modellieren, Konfigurieren und Ausführen von Prozessvarianten ermöglicht.

Die vorliegende Arbeit diskutiert die Weiterentwicklung und technische Umsetzung der in Provop entwickelten Konzepte. Dazu wird in dieser Arbeit der Provop-Lösungsansatz vorgestellt und verwandte Arbeiten betrachtet. Darauf aufbauend werden die Anforderungen zur Ausführung von Prozessvarianten in einem Workflow-Management-System (WfMS) diskutiert und verschiedene Konzepte zu deren Umsetzung vorgestellt. Abschließend wird die prototypische Umsetzung gezeigt.

Danksagung

Ich möchte mich in erster Linie bei Manfred Reichert und Alena Hallerbach für die gute Betreuung meiner Diplomarbeit bedanken. Aufgrund der regen Kommunikation mit meinen Betreuern war ich in der Lage die vorliegende Arbeit in einer ansprechenden Art und Weise zu verfassen. Des Weiteren möchte ich mich bei der Daimler AG und im Speziellen bei der Abteilung Daten- und Prozessmanagement des Forschungszentrums Ulm bedanken, dass ich dort im Rahmen einer Diplomandenstelle die vorliegende Arbeit erstellen konnte. Zusätzlich danke ich meinen Korrekturlesern, denen auch die kleinsten Fehler aufgefallen sind. Ganz besonders möchte ich mich bei meinen Eltern bedanken, dass sie mich während meiner gesamten Studienzeit unterstützt und mir somit einen akademischen Abschluss ermöglicht haben.

Inhaltsverzeichnis

1 Einführung

Bei der Modellierung von Prozessen hat man es in der Praxis oftmals mit zahlreichen Prozessvarianten zu tun. Jede Prozessvariante stellt dabei eine Anpassung eines Prozesses an bestimmte Rahmenbedingungen, wie z.B. organisatorische Zuständigkeiten innerhalb eines Unternehmens, dar. Diese Rahmenbedingungen kann man prinzipiell als den Kontext eines Prozesses verstehen. Die Erstellung von Prozessvarianten wird von heutigen Prozessmodellierungswerkzeugen nicht angemessen unterstützt. So können Prozessvarianten oftmals nur in separaten Modellen erstellt werden. Probleme daraus sind bspw., dass bei grundlegenden Änderungen u.U. mehrere Prozessvarianten betroffen sind und diese dann einzeln angepasst werden müssen. Entsprechend groß ist der Wartungs- und Pflegeaufwand bei einer hohen Anzahl an betroffenen Prozessvarianten.

Diese Problematik wird bei der Daimler AG durch das Forschungsprojekt Provop adressiert. Ziel von Provop ist es, einen generischen Lösungsansatz zu entwickeln, der ein transparentes und kontextabhängiges Management von Prozessvarianten erlaubt. Darüber hinaus werden Prozessvarianten von ihrer Modellierung über ihre Konfiguration und Ausführung bis hin zu ihrer Evolution durchgängig unterstützt.

Thema der vorliegenden Diplomarbeit ist die Weiterentwicklung und technische Umsetzung der in Provop entwickelten Konzepte zur Ausführung von variantenbehafteten Prozessen. Dabei wird untersucht, wie Provop in existierenden Workflow-Management-Systemen (WfMS) abgebildet werden kann.

1.1 Problembeschreibung

Inhalt der Diplomarbeit ist die Erfassung und Untersuchung existierender Ansätze zur Abbildung von Prozessvarianten in einem WfMS. Es wird untersucht, wie die Provop-Konzepte

zur kontextabhängigen Konfiguration von Prozessvarianten abgebildet werden können. Darüber hinaus wird betrachtet, wie Änderungen am Kontext während der Ausführung eines Prozesses zu Änderungen einer bestimmten Prozessvariante führen können. Des Weiteren wird untersucht, wie die Einhaltung von strukturellen und semantischen Abhängigkeiten gewährleistet werden kann. Die entwickelten Konzepte werden abschließend prototypisch umgesetzt.

1.2 Aufbau der Arbeit

Kapitel 2 führt grundlegende Begriffe aus dem Bereich des Prozessmanagements ein und stellt die wichtigsten Prozessbeschreibungssprachen zur Definition und Modellierung von Prozessmodellen vor. Kapitel 3 beschreibt den Provop-Lösungsansatz für ein transparentes und kontextabhängiges Management von Prozessvarianten. In Kapitel 4 werden verwandte Arbeiten aus Wissenschaft und Praxis erläutert. Kapitel 5 diskutiert die Anforderungen zur Abbildung der Konzepte des Provop-Lösungsansatzes in einem WfMS. In Kapitel 6 werden verschiedene Abbildungsmöglichkeiten für die Anforderungen beschrieben und miteinander verglichen. Kapitel 7 erläutert, wie die konkrete Umsetzung des Provop-Lösungsansatzes zur Ausführung von Prozessvarianten im WebSphere Integration Developer von IBM erfolgen kann und stellt Anwendungsbeispiele auf Basis des Prototypen vor. Kapitel 8 fasst die Arbeit zusammen.

2 Grundlagen

Nachdem in Kapitel 1 die Motivation und Problematik dieser Arbeit vorgestellt wurde, führt dieses Kapitel Grundlagen zum Verständnis dieser Arbeit ein. Abschnitt 2.1 erklärt Begriffe aus dem Bereich des Prozessmanagements. Abschnitt 2.2 gibt einen Überblick über Sprachen für die Beschreibung von ausführbaren Geschäftsprozessen.

2.1 Begrifflichkeiten

Prozess. Ein Prozess ist ein allgemeiner Ablauf mehrerer Abschnitte, bei denen es sich um Aufgaben, Ausführungen, Arbeitsschritte o.Ä. handeln kann. Zwischen diesen Prozessabschnitten bestehen bestimmte Abhängigkeiten, wie bspw. dass eine Aufgabe vor allen weiteren Aufgaben erledigt werden muss. Ein typischer Prozess umfasst folgende Elementtypen: Startereignis, Aktivität, Zerlegung, Sequenz, Auswahl, Parallelität und Abschlussereignis [RvS04].

Prozessmodell. Ein Prozessmodell beschreibt die Struktur eines Prozesses. Es bestimmt alle möglichen Pfade entlang des Prozesses und die Regeln für die Wahl der Pfade sowie weiterhin alle Aktivitäten, die ausgeführt werden müssen (vgl. Beispiel 2.1). Synonyme: Prozessdefinition, Prozess-Schema [RvS04].

Beispiel 2.1 (Prozessmodell) Abbildung 2.1 zeigt ein Beispiel für ein Prozessmodell. Es beschreibt in vereinfachter Form den Prozess einer Fahrzeugreparatur in einer Kfz-Werkstatt. Der Prozess beginnt mit der Annahme des Fahrzeugs. Daraufhin erfolgt eine Diagnose des Schadens. Wird hierbei ein Schaden erkannt, so wird das Fahrzeug repariert. Wird kein Schaden erkannt, z.B. im Falle eines Bedienfehlers, so entfällt diese Aktivität. Zum Schluss erfolgt die Übergabe des Fahrzeugs an den Kunden. Im Folgenden wird dieser Prozess als *Werkstattprozess* bezeichnet.

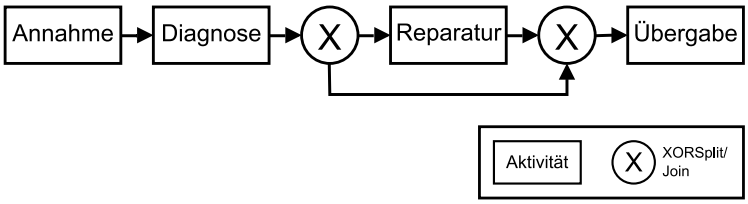

Abbildung 2.1: Vereinfachtes Prozessmodell einer Fahrzeugreparatur

Prozessfragment. Ein Prozessfragment beschreibt eine zusammenhängende Teilmenge von Elementen eines Prozesses (vgl. Beispiel 2.2). Es besteht aus denselben Elementtypen wie ein Prozessmodell.

Beispiel 2.2 (Prozessfragment) Die Aktivitäten der Annahme und jene der Diagnose stellen bspw. ein zusammenhängendes Prozessfragment des Werkstattprozesses in Abbildung 2.1 dar.

Prozessvariante. Eine Prozessvariante stellt eine Anpassung bzw. Modifizierung eines Prozesses dar. Sie entsteht, da Prozesse unter verschiedenen Rahmenbedingungen ausgeführt werden und hierfür Abweichungen vom standardisierten Ablauf nötig sind [HBR08a].

Prozessfamilie. Eine Prozessfamilie gruppiert die unterschiedlichen Varianten eines Prozesses zu einer zusammengehörenden Gruppe von Prozessmodellen.

Workflow-Management-System (WfMS). Ein WfMS ist ein Softwarepaket zur Unterstützung des Entwurfes und der Ausführung von Geschäftsprozessen [RvS04].

Prozessinstanz. Prozessinstanzen sind mit Hilfe eines WfMS ausführbare Prozesse. Sie basieren auf einem Prozessmodell und werden mit konkreten Daten angereichert, wie z.B. den Ausführungszuständen von Aktivitäten [RvS04].

Prozessadaption. Unter Prozessadaption versteht man das Anpassen bzw. Modifizieren eines Prozesses. Dabei unterscheidet man zwischen der Adaption einer einzelnen Prozessinstanz und der Adaption des Prozessmodells, der sog. Schema-Evolution [WRRM08].

Prozessbeschreibungssprache. Mit einer Prozessbeschreibungssprache wird graphisch oder textuell der Ablauf eines Prozesses beschrieben. Graphische Prozessbeschreibungs-

sprachen dienen zumeist nur der Prozessmodellierung, während textuelle Prozessbeschreibungssprachen für die Ausführung von Prozessmodellen in einem WfMS verwendet werden.

Prozesslebenszyklus. Der Prozesslebenszyklus besteht aus der Modellierung, Konfiguration und Ausführung von Prozessen. Aufgrund der ständigen Wartung und Optimierung von Prozessen stellt der Prozesslebenszyklus einen Kreislauf dar. Bei der Modellierung werden für die Prozesse sog. Prozessmodelle erstellt. Diese werden in der Phase der Konfiguration varianten-spezifisch angepasst. Bei der Ausführung werden die Prozesse z.B. rechnergestützt mit Hilfe eines WfMS ausgeführt [HBR09a].

2.2 Prozessbeschreibungssprachen

Im Folgenden wird ein Überblick der wichtigsten Prozessbeschreibungssprachen gegeben. Zu jeder Sprache wird deren Entwicklung und die grundlegende Funktionsweise erläutert. Zum direkten Vergleich der einzelnen Sprachen wird das Beispiel einer vereinfachten Reisebuchung verwendet (vgl. Beispiel 2.3), anhand dessen man den grundlegenden Aufbau einer Sprache erkennen kann. Hierbei ist zu beachten, dass einige Sprachen keine graphische Repräsentation vorsehen und dementsprechend ein Ausschnitt der textuellen Prozessbeschreibung abgebildet ist. Für ein tieferes Verständnis der einzelnen Sprachen wird auf die jeweilige Fachliteratur verwiesen.

Beispiel 2.3 (Vereinfachte Reisebuchung) Es sei folgender vereinfachte Prozess einer Reisebuchung gegeben: Zunächst erfolgt die Anmeldung beim Buchungssystem. Anschließend kann je nach Bedarf ein Flug, ein Hotel sowie ein Mietwagen gebucht werden. Nachdem alle Einzelbuchungen abgeschlossen sind, endet der Prozess mit der Bezahlung der Reise.

2.2.1 Business Process Modeling Notation

Die *Business Process Modeling Notation* (BPMN) ist eine graphische Modellierungssprache zur Beschreibung von Geschäftsprozessen. BPMN wurde von der Business Process Management Initiative (BPMI) entwickelt und wird seit 2005 von der Object Management

Group (OMG) verwaltet und weiterentwickelt. BPMN liegt aktuell in Version 1.2 [BPM09] vor; Version 2.0 befindet sich in der Entwicklung.

Die Modellierung von Prozessen erfolgt in *Business Process Diagrams (BPD)* (siehe Abbildung 2.2). Die BPMN-Modelle sind graph-orientiert. Sie bestehen aus sog. *Activities*, *Events* und *Gateways*. Activities beschreiben die Arbeitsschritte im Prozess und können sowohl atomar als auch ganze Prozesse sein. Events beschreiben Ereignisse während der Durchführung des Prozesses. Gateways dienen zur Strukturierung des Kontrollflusses. Der Kontrollfluss zwischen den Elementen wird durch den *Sequence Flow* modelliert. Der Nachrichtenaustausch zwischen Elementen erfolgt über den *Message Flow*. Für die Zuordnung von Zuständigkeiten und Unterteilung in funktionale Bereiche gibt es *Pools* und *Lanes*. Des Weiteren existieren *Artifacts*, mit denen die Prozessmodelle zu Dokumentationszwecken mit Informationen angereichert werden können. Diese haben aber keinen Einfluss auf die Struktur oder den Ablauf des Prozesses.

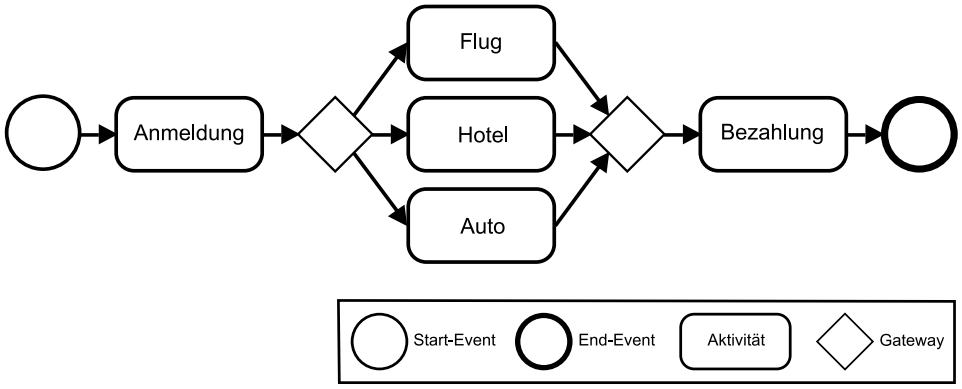

Abbildung 2.2: BPMN-Beispiel einer Reisebuchung

2.2.2 XML Process Definition Language

Die *XML Process Definition Language* (XPDL) ist eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen. XPDL wird seit 1993 von der Workflow Management Coalition (WfMC) entwickelt und liegt zurzeit in Version 2.1 vor [WfM08]. Durch die XPDL soll der Austausch von Prozessmodellen zwischen verschiedenen WfMS vereinfacht werden.

XPDL spezifiziert Prozessmodelle, wie in Abbildung 2.3 dargestellt, durch eine XML-Datei. Ein XPDL-Prozess besteht generell aus *Activities* und *Transitions*. Activities können hierbei

Aktivitäten, eigene Prozesse sowie Strukturknoten sein. Die Transitions spezifizieren den Kontrollfluss zwischen den Activities. Für die Beschreibung von Prozessen mittels XPDL ist keine graphische Repräsentation vorgesehen, die Sprachelemente können aber mittels BPMN graphisch dargestellt werden. Mit XPDL 2.1 lassen sich alle Spezifizierungen von BPMN 1.1 abbilden.

```
<WorkflowProcess Id="wp1" Name="Reisebuchung">
 <Activities>
  <Activity Id="act1" Name="Anmeldung">
   <Implementation>
    <No/>
   </Implementation>
   <TransitionRestrictions>
    <TransitionRestriction>
     <Split Type="OR">
      <TransitionRefs>
       <TransitionRef Id="t1"/><TransitionRef Id="t2"/><TransitionRef Id="t3"/>
      </TransitionRefs>
     </Split>
    </TransitionRestriction>
   </TransitionRestrictions>
  </Activity>
[...]
```

Abbildung 2.3: XPDL-Ausschnitt einer Reisebuchung

2.2.3 Business Process Execution Language

Die *Business Process Execution Language* (BPEL) ist ebenfalls eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen im Bereich Web Services. BPEL wurde als Vereinigung der Sprachen XLANG und WSFL von IBM, BEA und Microsoft entwickelt und wurde 2003 an die (OASIS) übergeben. BPEL liegt aktuell in der Version 2.0 vor [BPE07].

In BPEL werden Prozessmodelle durch eine XML-Datei spezifiziert (vgl. Abbildung 2.4). Ein BPEL-Prozess besteht aus *Basic Activities*, *Structured Activities* und *Scopes*. Basic Activities sind grundlegende atomare Aktivitäten eines Prozesses wie bspw. das Zuweisen einer Variable mittels *<assign>*. Structured Activities dienen zur Strukturierung des Kontrollflusses. Hierbei ist zu beachten, dass BPEL block-orientiert ist. Graph-orientierte Konstrukte können mit einem speziellen *<flow>*- oder *<link>*-Konstrukt modelliert werden, wobei Zyklen nur bedingt erlaubt sind. Scopes dienen zum Bündeln von Aktivitäten zu transaktionalen Einheiten. Dadurch kann die Sichtbarkeit von und der Zugriff auf Varia-

blen begrenzt werden. Der BPEL-Standard definiert keine graphische Darstellung für die Prozessmodelle.

```
<sequence>
 <receive>
 <invoke name="Anmeldung" />
 <flow>
  <links>
   <link name="t1" /><link name="t2" /><link name="t3" />
   <link name="t4" /><link name="t5" /><link name="t6" />
  </links>
  <empty>
   <sources>
    <source linkName="t1" /><source linkName="t2" />
    <source linkName="t3" />
   </sources>
  </empty>
  <flow>
   <targets><target linkName="t1"></targets>
   <sources><source linkName="t4"></sources>
   <invoke name="Flug">
  </flow>
[...]
```

Abbildung 2.4: BPEL-Ausschnitt einer Reisebuchung

2.2.4 Yet Another Workflow Language

Die Prozessbeschreibungssprache *Yet Another Workflow Language* (YAWL) wurde an der Eindhoven University of Technology sowie an der Queensland University of Technology entwickelt. Die Sprache wurde ausgehend von den Ergebnissen einer Analyse existierender WfMS und existierender Prozessbeschreibungssprachen sowie anhand der Workflow-Patterns [AHKB03] entwickelt. YAWL basiert auf Petri-Netzen, wurde aber um zusätzliche Mechanismen für die Modellierung von komplexeren Prozessmodellen erweitert [AH05].

Ein Workflow in YAWL besteht aus einer Menge von Extended-Workflow-Netzen (EWF-Netze). Ein EWF-Netz stellt die Prozessdefinition dar und besteht aus *Tasks* und *Bedingungen* (siehe Abbildung 2.5). Ein Task entspricht einer Transition, eine Bedingung einer Stelle bei Petri-Netzen. Jedes EWF-Netz hat genau eine Eingangs- sowie eine Ausgangsbedingung. Tasks können entweder atomar oder zusammengesetzt sein. Zusammengesetzte Tasks werden wiederum durch ein EWF-Netz spezifiziert. Zur Strukturierung gibt es

in YAWL die Strukturknoten AND-, XOR- und ORSplit sowie die jeweiligen Join-Konstrukte. Der Kontrollfluss wird wie bei Petri-Netzen durch Tokens gesteuert [AH05].

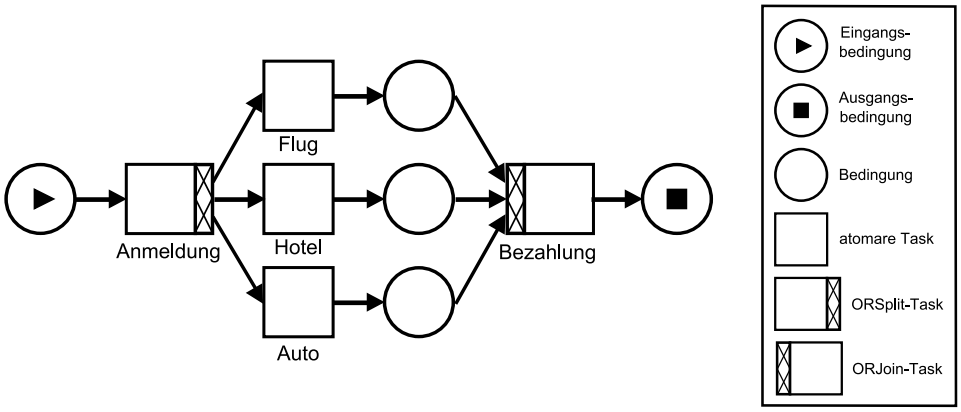

Abbildung 2.5: YAWL-Beispiel einer Reisebuchung (nach [AH05])

2.3 Zusammenfassung

In diesem Kapitel wird die Grundlage für ein besseres Verständnis der vorliegenden Arbeit geschaffen. Dazu werden zunächst wichtige Begriffe aus dem Bereich des Prozessmanagements eingeführt und erklärt.

Anschließend wird ein Überblick der wichtigsten Prozessbeschreibungssprachen gegeben. Zu allen Sprachen wird anhand eines Beispiels der grundlegende Aufbau der Sprache erklärt und somit ein Grundverständnis für die spätere Evaluation der Sprachen zur Abbildung der Provop-Konzepte geschaffen (vgl. Kapitel 6). Für die einzelnen Sprachen gibt es sowohl graphische (z.B. BPMN) als auch rein textuelle Repräsentationen (z.B. XPDL). Des Weiteren unterscheiden sich die Sprachen oftmals wesentlich in der Handhabung des Kontrollflusses.

Im folgenden Kapitel wird auf Basis der vorgestellten Grundlagen der Provop-Ansatz zum Management von Prozessvarianten beschrieben.

3 Provop-Lösungsansatz

Wie bereits motiviert, stellt das Management von Prozessvarianten eine große Herausforderung dar, der konventionelle Ansätze nicht gerecht werden. Diese erlauben nur das Ausmodellieren aller Prozessvarianten in separaten Modellen oder die Abbildung aller Varianten eines Prozesses in einem Prozessmodell. Bei einer Vielzahl von Prozessvarianten sind diese Ansätze nicht mehr ausreichend, da sie zu hohen Modellierungs- und Wartungsaufwänden führen. Im Rahmen des Projekts Provop wurde daher ein Ansatz entwickelt, der das transparente Management einer Vielzahl von Prozessvarianten ermöglicht. Dieses Kapitel beschreibt die konventionellen Ansätze und stellt anschließend den Provop-Lösungsansatz im Detail vor.

Das Kapitel gliedert sich wie folgt: Abschnitt 3.1 erläutert, wie heute das Management von Prozessvarianten realisiert wird. Abschnitt 3.2 stellt die Grundidee von Provop vor. Abschnitt 3.3 erklärt, wie Provop den Anwender bei der Modellierung von Prozessvarianten unterstützt. Abschnitt 3.4 erklärt die Konzepte für die Konfiguration von Prozessvarianten. In Abschnitt 3.5 wird auf die Ausführung von Prozessvarianten eingegangen.

3.1 Konventionelles Variantenmanagement

Zurzeit existieren zwei konventionelle Ansätze für die Modellierung von Prozessvarianten. Der erste Ansatz verfolgt das Ausmodellieren aller Prozessvarianten in eigenen Prozessmodellen und wird im Folgenden als *Mehr-Modell-Ansatz* bezeichnet. Beim zweiten Ansatz werden alle Prozessvarianten in einem Prozessmodell zusammengefasst. Dementsprechend wird dieser Ansatz als *Ein-Modell-Ansatz* bezeichnet. Im Folgenden werden diese Ansätze beschrieben und diskutiert.

3.1.1 Mehr-Modell-Ansatz

Beim Mehr-Modell-Ansatz wird jede Prozessvariante ausmodelliert. Somit wird jede Prozessvariante durch ein separates Prozessmodell repräsentiert. Bei der Erstellung einer Prozessvariante kann in diesem Ansatz auf existierende Prozessmodelle zurückgegriffen werden. Eine Prozessvariante kann z.B. als Kopie erzeugt und anschließend an die gegebenen Rahmenbedingungen angepasst werden. Um die Zugehörigkeit von solchen Prozessvarianten zu einem Prozesstyp auszudrücken, werden diese ähnlich benannt oder in gemeinsamen Verzeichnissen abgelegt (vgl. Abbildung 3.1).

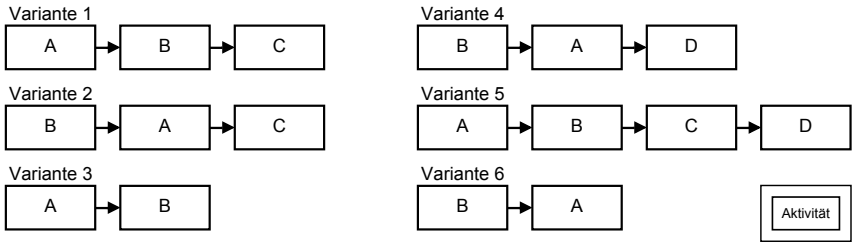

Abbildung 3.1: Mehr-Modell-Ansatz

Existiert eine Vielzahl von Prozessvarianten, sind dementsprechend auch viele Prozessmodelle zu verwalten und zu pflegen. Aus der Ähnlichkeit der Varianten ergibt sich oft eine hohe Anzahl an redundanten Prozessfragmenten. Bei etwaigen Änderungen führt dies u.U. zu einem hohen Änderungsaufwand, da gegebenenfalls jedes einzelne Prozessmodell separat angepasst werden muss. Dieses Vorgehen ist zudem fehleranfällig.

Beim Mehr-Modell-Ansatz ist nicht klar erkennbar, welche Prozessfragmente variantenspezifisch sind. Die Abhängigkeit von gewissen Rahmenbedingungen lässt sich nur für das gesamte Prozessmodell, z.B. als Metadaten, angeben, nicht aber für einzelne Fragmente des Prozesses.

Soll eine Prozessvariante in einem WfMS ausgeführt werden, so wird das entsprechende Prozessmodell instanziiert. Ein Wechsel zwischen zwei Prozessvarianten zur Laufzeit, bspw. aufgrund veränderte Rahmenbedingungen, ist bei diesem Ansatz nur durch den Abbruch der aktuellen und Instanziierung einer neuen Prozessvariante möglich.[1]

[1] Einige Ansätze aus der Literatur, wie bspw. ADEPT [DRRM+09], bieten auch eine dynamische Migration auf das neue Prozessmodell zur Laufzeit an.

Existieren für einen Prozess viele Prozessvarianten, kann deren Handhabung schnell komplex und unübersichtlich werden. Der Mehr-Modell-Ansatz skaliert daher schlecht für eine hohe Anzahl an Prozessvarianten.

3.1.2 Ein-Modell-Ansatz

Beim Ein-Modell-Ansatz werden alle Prozessvarianten eines Prozesstyps durch ein Prozessmodell repräsentiert. Die Abbildung der variantenspezifischen Prozessfragmente erfolgt z.B. durch die Verwendung von bedingten Verzweigungen. Die Verzweigungsbedingungen können verwendet werden, um die Abhängigkeit von bestimmten Rahmenbedingungen zu beschreiben.

Bei diesem Ansatz werden die Prozessvarianten nicht explizit als solche gekennzeichnet. Dies führt dazu, dass der Modellierer nicht mehr erkennen kann, ob eine bedingte Verzweigung im Prozessmodell eine Prozessvariante spezifiziert oder eine normale alternative Verzweigung für alle Prozessvarianten.

Auch in diesem Ansatz können Modellredundanzen entstehen. Dies ist der Fall, wenn zwei Prozessvarianten eine unterschiedliche Reihenfolge der Aktivitäten aufweisen. Diese Aktivitäten werden für jede Prozessvariante an einer anderen Position im Prozessmodell und somit redundant modelliert (vgl. Aktivität A und B in Abbildung 3.2).

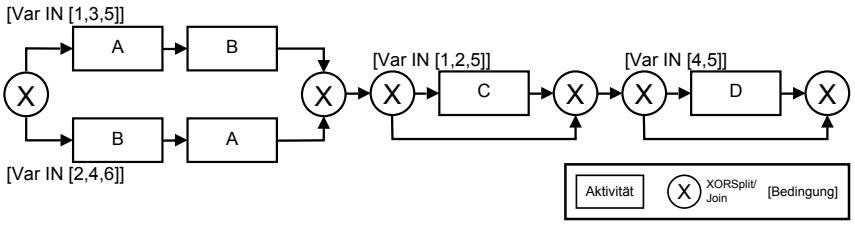

Abbildung 3.2: Ein-Modell-Ansatz

Für die Ausführung einer Prozessvariante wird das gesamte Prozessmodell instanziiert. Der Wechsel zwischen Prozessvarianten während der Ausführung stellt in diesem Ansatz kein Problem dar, da das Prozessmodell alle Prozessvarianten beinhaltet und somit der Wechsel zwischen Varianten implizit durch die Wahl eines anderen Ausführungspfades erfolgt.

Bei einer hohen Anzahl von Prozessvarianten wird das Prozessmodell sehr groß und unübersichtlich. Durch die Vielzahl an Kontrollfluss-Kanten kann ggf. ein inkorrektes Pro-

zessmodell mit unerlaubten Zyklen o.Ä. entstehen. Dementsprechend skaliert auch dieser Ansatz bei einer Vielzahl von Prozessvarianten schlecht.

3.1.3 Fazit

Beide Ansätze sind in der Praxis gängige Techniken für das Management von Prozessvarianten in heutigen Prozessmanagement-Werkzeugen. Dies erklärt sich dadurch, dass beide mit den Möglichkeiten der Werkzeuge relativ leicht umgesetzt werden können. Allerdings ist die Unterstützung für das Variantenmanagement bei einer hohen Anzahl an Prozessvarianten unzureichend. So können in beiden Ansätzen Modellredundanzen auftreten. Die Aufwände für die Pflege und Wartung der Prozessmodelle erhöhen sich dadurch und werden u.U. sehr umfangreich. Daraus resultiert eine schlechte Skalierbarkeit der Ansätze. Des Weiteren mangelt es beiden Ansätzen an Transparenz über die Prozessvarianten. Die Unterschiede der Prozessvarianten sind entweder in separaten Modellen oder in der Ablauflogik, d.h. in bedingten Verzweigungen, versteckt. Auch die Abhängigkeit von bestimmten Rahmenbedingungen wird in beiden Ansätzen nicht ausreichend berücksichtigt.

3.2 Grundidee von Provop

Provop basiert auf der Beobachtung, dass Prozessvarianten üblicherweise nicht neu erstellt werden, sondern auf existierenden Modellen eines Prozesstyps aufbauen. Prozessvarianten können durch ein Ausgangsmodell und variantenspezifische Abweichungen beschrieben werden. Diese Vorgehensweise kann im Prinzip zur Ableitung aller Prozessvarianten eines Prozesstyps verwendet werden. Die Grundidee von Provop ist daher, mit Hilfe eines sog. Basisprozesses und zugehörigen Änderungsoperationen, d.h. variantenspezifischen Abweichungen, alle relevanten Prozessvarianten abzuleiten (vgl. Beispiel 3.1).

Beispiel 3.1 (Grundidee von Provop) Abbildung 3.3 beschreibt die Grundidee von Provop. Für einen Basisprozess aus der Werkstattdomäne (a) werden zwei variantenspezifische Abweichungen (b) spezifiziert. Hierbei soll die Prüfung entfernt und eine Wartung eingefügt werden. Daraus wird die entsprechende Prozessvariante (c) abgeleitet.

Eine wesentliche Anforderung an den Ansatz stellt die Durchgängigkeit über alle Phasen des Prozesslebenszyklus dar. Entsprechend beschreiben die nachfolgenden Abschnit-

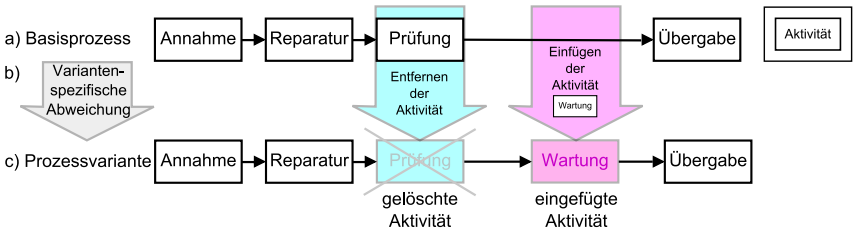

Abbildung 3.3: Ableitung einer Prozessvariante auf Basis eines Ausgangsmodells

te 3.3 bis 3.5 wie Provop die Phasen der Modellierung, Konfiguration und Ausführung von Prozessvarianten unterstützt [HBR08b].

3.3 Modellierung von Prozessvarianten

Die erste Phase des Prozesslebenszyklus stellt die Modellierung einer Prozessfamilie dar. In dieser Phase werden die Prozesse mittels eines Prozessmodells graphisch erstellt. In Provop ist es dem Modellierer möglich, ein Ausgangsmodell zu erstellen, welches durch die Angabe von variantenspezifischen Abweichungen zu einer Prozessvariante konfiguriert wird. Dabei ist es möglich, die von den Änderungen betroffenen Fragmente im Prozessmodell eindeutig zu spezifizieren und zu referenzieren. Für die Beschreibung der variantenspezifischen Abweichungen werden dem Modellierer höherwertige Änderungsoperationen zur Verfügung gestellt. Diese Änderungsoperationen werden dann auf das Ausgangsmodell angewandt und daraus die Prozessvariante erstellt. Anhand der Änderungsoperationen sind alle praxisrelevanten Anwendungsfälle zur Modellierung von Prozessvarianten abgedeckt. Des Weiteren ist es dem Modellierer möglich, die Änderungsoperationen zu gruppieren. Dies erleichtert die Beschreibung von komplexen Abweichungen und macht diese transparenter [HBR08a].

3.3.1 Basisprozess und Referenzpunkte

Das Ausgangsmodell für die Beschreibung von Prozessvarianten wird im Folgenden als *Basisprozess* bezeichnet. Um im Basisprozess explizit kennzeichnen zu können, welche Prozessfragmente geändert werden können, werden sog. *Aufsetzpunkte* verwendet (vgl.

Beispiel 3.2). Die Platzierung der Aufsetzpunkte erfolgt an den Knotenein- und -ausgängen von Aktivitäten und Strukturknoten im Basisprozess [HBR09a].

Beispiel 3.2 (Basisprozess mit Aufsetzpunkten) Abbildung 3.4 zeigt den Basisprozess aus der Werkstattdomäne. Im Prozessmodell werden die zwei Stellen D.out und Ü.in explizit durch Aufsetzpunkte gekennzeichnet. Diese sind das Ende der Diagnose sowie der Beginn der Übergabe des Fahrzeugs an den Kunden. Änderungsoperationen können diese Aufsetzpunkte nun direkt referenzieren.

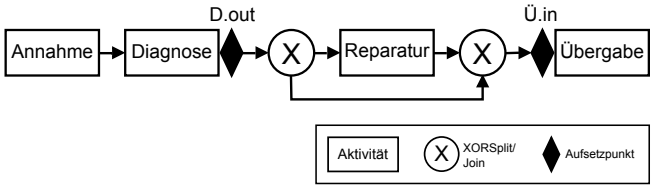

Abbildung 3.4: Basisprozess mit Aufsetzpunkten an Knotenein- und Knotenausgängen

3.3.2 Änderungsoperationen

Für die Ableitung einer Prozessvariante aus einem Basisprozess, müssen die variantenspezifischen Abweichungen beschrieben werden. In Provop erfolgt dies durch die Spezifizierung von höherwertigen Änderungsoperationen. Diese sind *Einfügen*, *Löschen* und *Verschieben von Prozessfragmenten*, sowie das *Ändern von Attributwerten* (vgl. [HBR08b]). Im Folgenden werden die einzelnen Änderungsoperationen kurz beschrieben.

Einfügen mittels Insert-Operation. Zur Erweiterung des Basisprozesses kann ein beliebiges Prozessfragment eingefügt werden (vgl. Beispiel 3.3). Die Position, an der eingefügt werden soll, wird durch die Aufsetzpunkte im Basisprozess beschrieben. Das Einfügen der Prozessfragmente erfolgt immer parallel zu den im Basisprozess referenzierten Prozessfragmenten.

Beispiel 3.3 (Einfügen eines Prozessfragments) Abbildung 3.5 beschreibt das Einfügen eines Prozessfragments mittels einer höherwertigen Änderungsoperation. In den in (a) dargestellten Basisprozess des Werkstattprozesses soll vor der Übergabe des Fahrzeugs an den Kunden eine zusätzliche Prüfung eingefügt werden. Dies wird durch eine Insert-Operation beschrieben (b). Die Parameter S und E referenzieren auf die im Basisprozess

dargestellten Aufsetzpunkte X und Y. Durch das Einfügen des neuen Prozessfragments ergibt sich die dargestellte Prozessvariante (c).

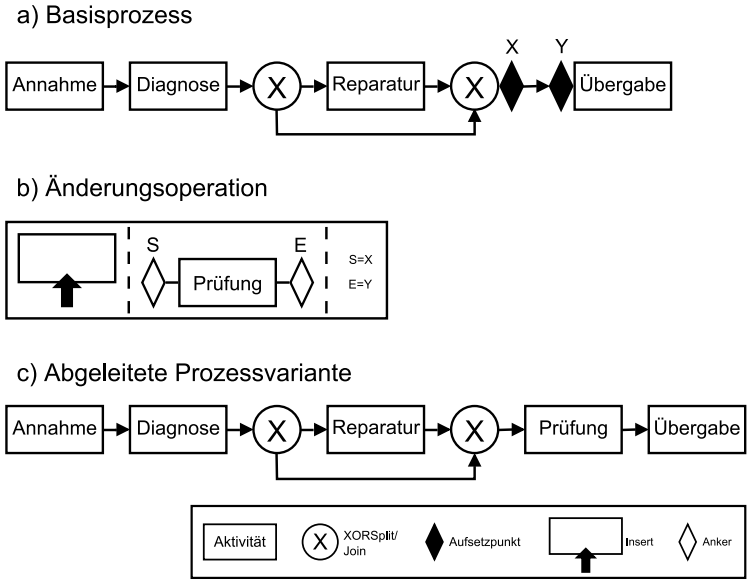

Abbildung 3.5: a) Basisprozess mit einer b) Insert-Änderungsoperation und c) der daraus abgeleiteten Prozessvariante

Löschen mittels Delete-Operation. Durch die Verwendung der Delete-Operation werden Prozesselemente bzw. ganze Prozessfragmente aus dem Basisprozess entfernt (vgl. Beispiel 3.4). Ein einzelnes Prozesselement wird durch die Angabe seiner eindeutigen Prozesselement-ID gelöscht (vgl. Abbildung 3.6). Zum Löschen von Prozessfragmenten wird durch die Angabe von Aufsetzpunkten ein Prozessfragment des Basisprozesses referenziert und alle Prozesselemente entfernt, die sich zwischen den spezifizierten Aufsetzpunkten befinden [HBR08b].

Beispiel 3.4 (Löschen eines Prozessfragments) Abbildung 3.6 zeigt das Löschen eines Prozessfragments mittels einer höherwertigen Änderungsoperation. Im Basisprozess (a) soll die Wartung des Fahrzeugs aus dem Prozessmodell entfernt werden. Dies wird durch die eine Delete-Operation beschrieben (b). Die Operation bezieht sich hier auf ein einzelnes Prozesselement. Durch das Löschen des Prozessfragments ergibt sich die dargestellte Prozessvariante (c).

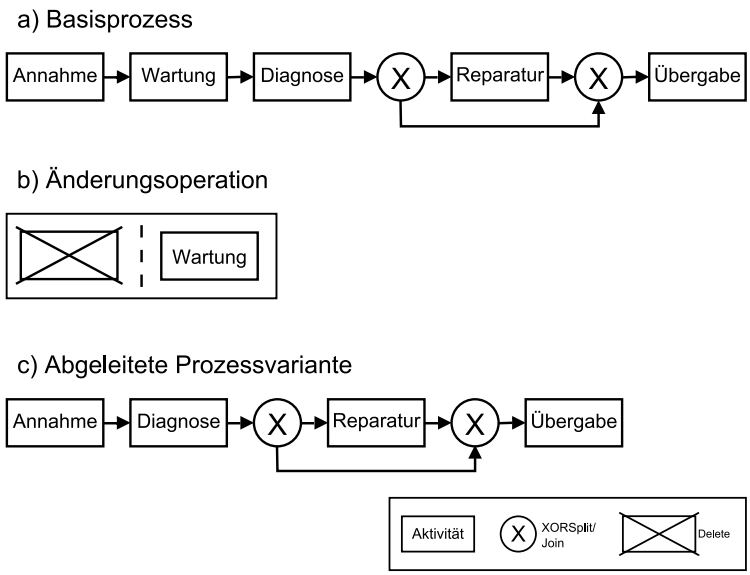

Abbildung 3.6: a) Basisprozess mit b) einer Delete-Änderungsoperation und c) der daraus abgeleiteten Prozessvariante

Verschieben mittels Move-Operation. Mit Hilfe der Move-Operation können Prozessfragmente im Basisprozess verschoben werden (vgl. Beispiel 3.5). Diese Operation setzt sich im Prinzip aus einer Delete- und einer Insert-Operation zusammen. Zunächst wird das zu verschiebende Prozessfragment mittels Aufsetzpunkten spezifiziert und an der ursprünglichen Position entfernt. Anschließend erfolgt ein Einfügen des Prozessfragments analog zur Insert-Operation, wobei die neue Zielposition ebenfalls durch Aufsetzpunkte des Basisprozesses angegeben wird.

Beispiel 3.5 (Verschieben eines Prozessfragments) Abbildung 3.7 zeigt das Verschieben eines Prozessfragments mittels einer höherwertigen Änderungsoperation. Im Basisprozess (a) soll die Wartung des Fahrzeugs erst vor der Übergabe stattfinden und nicht schon vor der Diagnose. Dieses Verschieben der Wartung wird durch eine Move-Operation beschrieben (b). Die Referenzierung erfolgt über die Aufsetzpunkte im Basisprozess. Durch das Verschieben des Prozessfragments ergibt sich die dargestellte Prozessvariante (c).

Ändern von Attributwerten mittels Modify-Operation. Die Modify-Operation erlaubt das Anpassen von Attributwerten von Prozesselementen (vgl. Beispiel 3.6). Die Identifizierung der zu ändernden Prozesselemente erfolgt über ihre Prozesselement-ID. Des Weiteren muss der eindeutige Attributbezeichner und ein korrekter Attributwert angegeben werden,

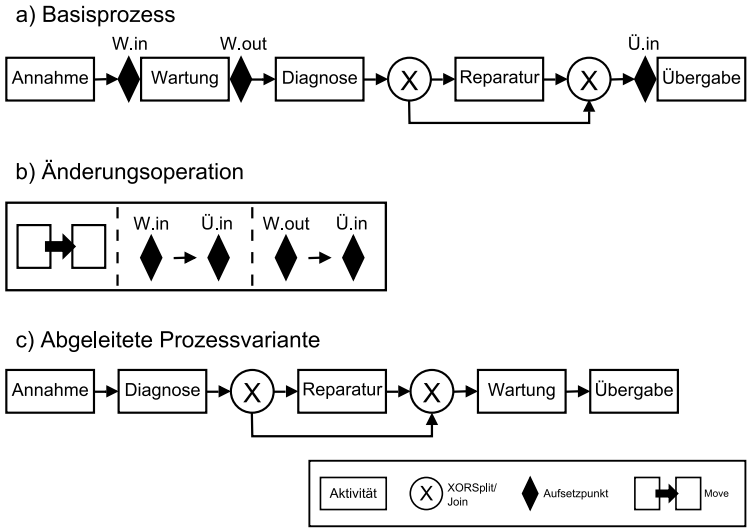

Abbildung 3.7: a) Basisprozess mit b) einer Move-Änderungsoperation und c) der daraus abgeleiteten Prozessvariante

der sich im Wertebereich des Attributtyps eines Elements befindet. Einige Attributwerte, wie z.B. die Prozesselement-ID, dürfen nicht durch die Modify-Operation verändert werden und werden daher als fix parametrisiert.

Beispiel 3.6 (Ändern von Attributwerten) Abbildung 3.8 zeigt das Ändern von Attributwerten mittels einer höherwertigen Änderungsoperation. Im Basisprozess (a) soll die Rolle des Bearbeiters für die Reparatur geändert werden. Dies wird durch eine Modify-Operation beschrieben (b). Der neue Wert des Attributs wird als Parameter der Änderungsoperation angegeben. Durch das Ändern des Attributwertes ergibt sich die dargestellte Prozessvariante (c).

3.3.3 Gruppieren der Änderungsoperationen

Bei der Spezifikation einer Prozessvariante werden meist mehrere Änderungsoperationen auf den Basisprozess angewandt (vgl. Beispiel 3.7). Um diese Zusammengehörigkeit abzubilden, werden Änderungsoperationen in Provop zu einem Objekt zu gruppiert. Solche Gruppen werden als *Optionen* bezeichnet. Für Optionen gilt das Prinzip der Atomarität. Dies bedeutet, dass immer alle zugehörigen Änderungsoperationen einer Option gemeinsam auf den Basisprozess angewandt werden müssen oder gar keine. Eine teilweise An-

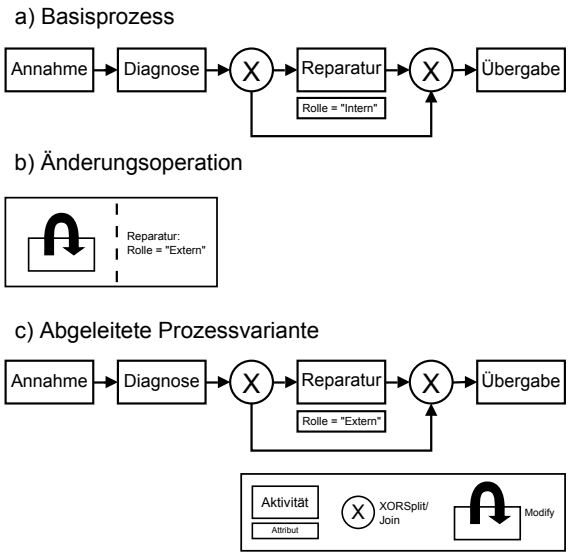

Abbildung 3.8: a) Basisprozess mit b) einer Modify-Änderungsoperation und c) der daraus abgeleiteten Prozessvariante

wendung ist nicht erlaubt. Die Änderungsoperationen werden in den Optionen als Sequenz gespeichert [HBR09a].

Beispiel 3.7 (Gruppieren von Änderungsoperationen) Hat die Werkstatt eine hohe Auslastung und ist nicht mehr in der Lage alle Reparaturen termingerecht zu erledigen, so werden einzelne Reparaturen an eine andere Werkstatt ausgelagert. Die Aktivität der Reparatur wird somit von einem externen Bearbeiter erledigt. Zudem muss vor der externen Reparatur das Fahrzeug in die Partner-Werkstatt gebracht und anschließend wieder abgeholt werden. Das Einfügen dieser beiden Aktivitäten ist aber nur bei einer Reparatur durch eine andere Werkstatt erwünscht. Dementsprechend müssen diese Änderungsoperationen zu einer Option zusammengefasst werden.

3.4 Konfiguration von Prozessvarianten

Bei der Konfiguration von Prozessvarianten wählt der Anwender aus, welche der modellierten Optionen auf den Basisprozess angewandt werden sollen, um eine spezifische Prozessvariante zu generieren. Die Auswahl der Optionen hängt vom variantenspezifischen Kontext ab, d.h. von den speziellen Rahmenbedingungen für die eine Prozessvariante er-

forderlich ist. In Provop wird der Anwender bei dieser kontextabhängigen Auswahl unterstützt [HBR09a].

3.4.1 Beschreibung des Kontexts

Unter dem Kontext einer Prozessvariante versteht man die Informationen und Rahmenbedingungen, die für die Auswahl einer spezifischen Prozessvariante relevant sind. Um dem Anwender die Auswahl der Optionen für die Konfiguration einer Variante zu erleichtern, muss der Kontext der Prozessvarianten berücksichtigt werden können. Dazu muss der Kontext zunächst in einer maschinenlesbaren Form mittels eines *Kontextmodells* erfasst und beschrieben werden können. In Provop wird der Kontext dazu durch *Kontextvariablen* spezifiziert. Ein konkreter Kontext wird durch entsprechende *Kontextwerte* aus dem Wertebereich der Kontextvariable beschrieben.

Die kontextbasierte Konfiguration der Prozessvarianten erfolgt durch die Zuweisung von sog. *Kontextbedingungen* zu Optionen bzw. deren Auswertung hinsichtlich des konkreten Kontextes des Prozesses (vgl. Beispiel 3.8). Ist die Kontextbedingung erfüllt, wird die Option auf den Basisprozess angewandt, andernfalls wird sie ausgelassen [HBR09a].

Beispiel 3.8 (Beschreibung des Kontexts) Ein Beispiel für einen Kontext stellt der Kostenvoranschlag für die Reparatur dar, welcher während der Aktivität Diagnose erstellt wird. Dementsprechend gibt es im Kontextmodell eine Kontextvariable `Kostenvoranschlag` mit möglichen Werten aus den Intervallen $>$ `800` € und $\leq$ `800` €. Bei einem Kostenvoranschlag von $>$ `800` € soll nach der Diagnose beim Kunden nachgefragt werden, ob er die Reparatur durchführen lassen will. Dementsprechend muss die Aktivität Rückfrage in das Prozessmodell eingefügt werden. Die Option, welche die Rückfrage einfügt, bekommt die Kontextbedingung `Kostenvoranschlag` $>$ `800` € zugewiesen (vgl. Abbildung 3.9 (b)).

3.4.2 Beziehungen zwischen den Optionen

Bei der Konfiguration der Prozessvarianten muss darauf geachtet werden, dass die Optionen strukturell und semantisch kompatibel sind. Es darf nicht vorkommen, dass widersprüchliche Optionen gemeinsam auf denselben Basisprozess angewandt werden. Um die Abhängigkeiten zwischen Optionen ausdrücken zu können, werden in Provop explizite Optionsbeziehungen angeboten, welche zwischen den Optionen modelliert werden können

(vgl. Beispiel 3.9). Die unterstützten Optionsbeziehungen sind *Implikation*, *wechselseitige Implikation*, *wechselseitiger Ausschluss*, *Auswahl n-aus-m* und *Hierarchie* [HBR09a].

Die **Implikation** koppelt Optionen aneinander. Impliziert Option A die Option B, so muss bei Anwendung von Option A auch Option B zur Bildung der Prozessvariante angewandt werden. Die Implikation ist unidirektional.

Die **wechselseitige Implikation** verhält sich analog zur Implikation, ist aber bidirektional und lässt sich somit auch durch zwei Implikationen darstellen. Implizieren sich die Optionen A und B wechselseitig, so muss bei Anwendung von Option A auch Option B sowie umgekehrt bei Anwendung von Option B auch Option A angewandt werden.

Der **wechselseitige Ausschluss** verhindert, dass Optionen gemeinsam angewandt werden. Schließen sich Option A und Option B wechselseitig aus, so darf bei Anwendung von Option A die Option B nicht angewandt werden und umgekehrt.

Anhand der **Auswahl n-aus-m** wird die Auswahl und Anwendung einer gewissen Teilmenge aus einer Menge von Optionen vorgegeben. Sollen bspw. genau 2-aus-4 Optionen angewandt werden, so darf nach der Anwendung von 2 Optionen keine weitere Option mehr angewandt werden. Diese Auswahl schließt auch aus, dass weniger als zwei Optionen angewandt werden.

Beispiel 3.9 (Optionsbeziehung) Abbildung 3.9 beschreibt ein Beispiel für eine Abhängigkeit zwischen zwei Optionen. Zunächst wird der Basisprozess der Werkstatt dargestellt (a). Für diesen Basisprozess existieren zwei Optionen (b). Option 1 fügt eine Rückfrage beim Kunden nach der Diagnose ein. Diese Option soll angewandt werden, wenn der Kostenvoranschlag $>$ 800 € Reparaturkosten vorsieht. Option 2 lagert die Reparatur an eine externe Werkstatt aus (Ändern der Rolle des Bearbeiters der Aktivität) und fügt zusätzlich einen Transport zur und eine Abholung von der externen Werkstatt ein. Diese Option soll angewandt werden, wenn die Auslastung der Werkstatt hoch ist, d.h. wenn die Werkstatt nur noch wenige Kapazitäten zur Reparatur der Fahrzeuge zur Verfügung hat.

Eine Abhängigkeit im Werkstattprozess ist bspw., dass bei einem hohen Kostenvoranschlag das Fahrzeug nicht von einer anderen Werkstatt repariert werden soll, da solche Aufträge für die eigene Werkstatt wirtschaftlicher sind. Somit schließt das Einfügen der Rückfrage beim Kunden das Auslagern der Reparatur an eine andere Werkstatt aus. Die Optionsbeziehung (c) drückt dies aus. Die Prozessvariante, die nach der Anwendung von Option 1 und Option 2 resultiert, ist nicht Teil der Prozessfamilie (d).

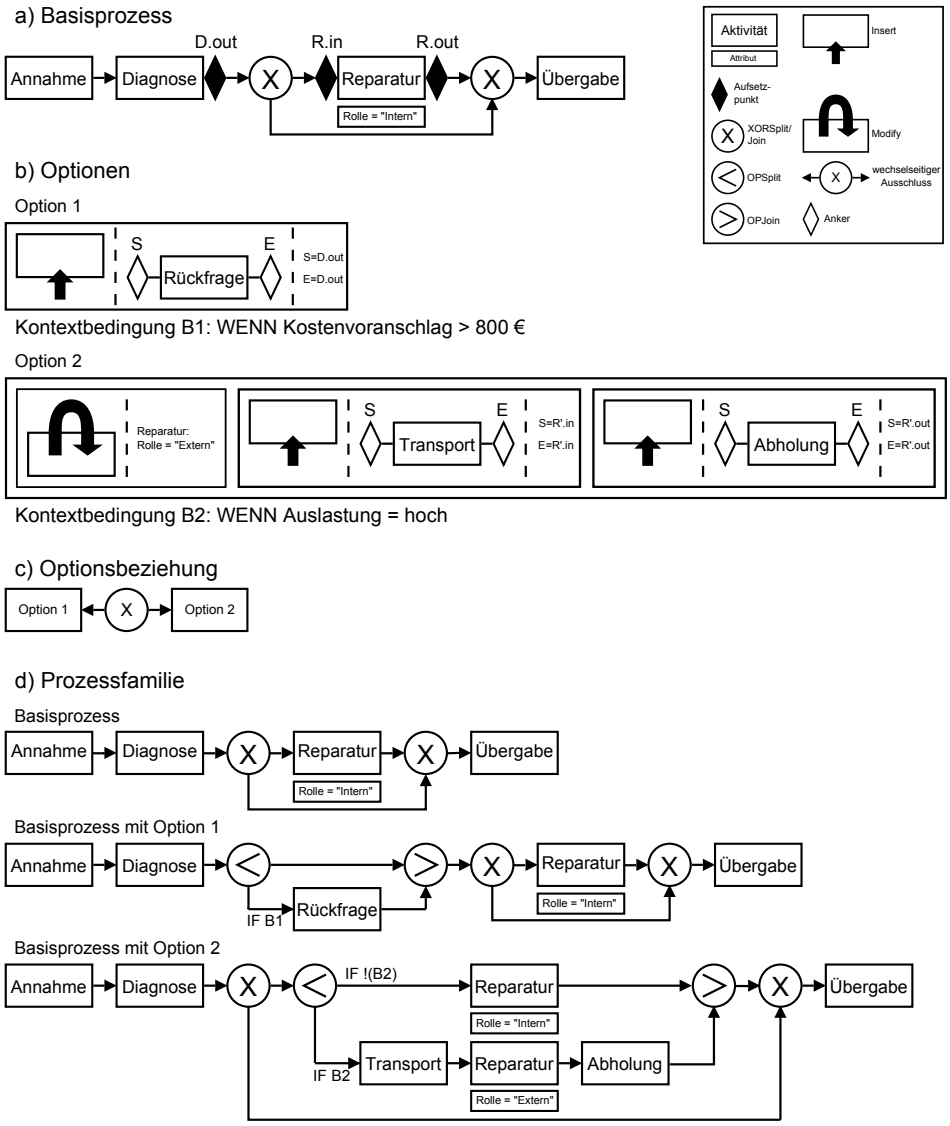

Abbildung 3.9: a) Basisprozess mit b) Optionen, c) einer Optionsbeziehung sowie d) der daraus resultierenden Prozessfamilie

3.4.3 Dynamische Konfiguration

Bei der Konfiguration von Prozessvarianten kann es vorkommen, dass gewisse Kontextwerte für die Auswahl der Optionen erst zur Laufzeit feststehen oder sich dynamisch ändern können. Dementsprechend muss es möglich sein, den Kontext dynamisch zur Laufzeit auszuwerten. Hierfür wird im Kontextmodell zwischen *statischen* und *dynamischen* Kontextvariablen unterschieden. Statisch bedeutet, dass der Wert der Kontextvariable einmal festgelegt wird und sich anschließend nicht mehr ändern kann. Dies geschieht während der Konfiguration. Dynamisch hingegen bedeutet, dass der Wert der Kontextvariable entweder während der Konfiguration oder während der Ausführung festgelegt wird und jederzeit geändert werden kann [HBR09a].

Aufgrund der dynamischen Kontextvariablen müssen die Optionen auch dynamisch zur Laufzeit angewandt werden können. Die Funktionsweise der einzelnen Änderungsoperation ist dieselbe wie in Abschnitt 3.3.2 beschrieben. Allerdings wird vor der Ausführung der dynamischen Prozessfragmente zur Laufzeit der Kontext ermittelt, die Kontextbedingung der Option ausgewertet und entschieden, ob die Option dynamisch angewandt werden soll [HBR09a]. Zusätzlich wird vor der Ausführung der dynamischen Prozessfragmente die Einhaltung der Optionsbeziehungen geprüft. Um zwischen Kontextbedingungen der Optionen und Optionsbeziehungen priorisieren zu können, wird zwischen *harten* und *weichen* Optionsbeziehungen unterschieden. Weiche Optionsbeziehungen sind nur während der statischen Konfiguration der Prozessvarianten relevant und werden bei der dynamischen Auswahl nicht berücksichtigt. Harte Optionsbeziehungen hingegen sind auch für die dynamische Auswahl relevant und werden höher priorisiert als die Kontextbedingungen. Wird bspw. durch die Anwendung einer dynamischen Option eine harte Optionsbeziehung verletzt, die Kontextbedingung der Option ist aber gültig, so wird die Option nicht angewandt. Ist die Kontextbedingung nicht erfüllt, aber eine harte Optionsbeziehung verlangt das Anwenden einer Option, so wird die Option angewandt [Hal10]. Auf diesen Aspekt wird in Kapitel 5 im Detail eingegangen.

3.5 Ausführung von Prozessvarianten

Vor der Ausführung von Prozessvarianten wird das fachlich modellierte und konfigurierte Prozessmodell in ein technisches Workflow-Modell überführt. Dieses kann dann in einem

WfMS ausgeführt werden. Provop fokussiert bei der Ausführung nicht auf ein generelles Übertragen des fachlichen Prozessmodells, sondern auf die Übertragung der relevanten Provop-Konstrukte [HBR09a].

3.5.1 Gewährleistung der Atomarität von Optionen

Ein wichtiger Aspekt bei der Ausführung ist die dynamische Anwendung der einzelnen Änderungsoperationen einer Option wie in Abschnitt 3.4.3 motiviert. Hier ist zu beachten, dass eine partielle Anwendung einer Option nicht erlaubt ist. Stattdessen wird eine Option als ein atomares Objekt betrachtet und dementsprechend ganz oder gar nicht angewandt. Dies bedeutet, falls bspw. eine Änderungsoperation einer Option dynamisch angewandt wurde und sich im Folgenden der Kontext so ändert, dass die Kontextbedingung der Option nicht mehr erfüllt ist, werden die verbleibenden Änderungsoperationen dieser Option trotzdem angewandt. Analog verhält es sich, wenn Änderungsoperationen aufgrund der Nichterfüllung der Kontextbedingung nicht angewandt werden konnten. In diesem Fall wird die gesamte Option nicht angewandt [HBR09a]. Dieses Vorgehen ist sinnvoll, da die Zusammengehörigkeit von einzelnen Änderungsoperationen durch die Gruppierung zu einer Option explizit angegeben wurde. Bei einer partiellen Anwendung der Optionen könnte diese Zusammengehörigkeit von Änderungsoperationen nicht mehr ausgedrückt werden.

3.5.2 Einhaltung der strukturellen und semantischen Abhängigkeiten

Wird bei der Ausführung die erste Änderungsoperation einer Option erreicht und wird mindestens eine harte Optionsbeziehung durch die mögliche Anwendung der Option verletzt, so wird die Änderungsoperation trotz gültigen Kontexts nicht angewandt. Dies resultiert aus der höheren Priorisierung harter Optionsbeziehungen (vgl Abschnitt 3.4.3). Somit wird auch zur Laufzeit die Einhaltung der strukturellen und semantischen Abhängigkeiten zwischen den Optionen gewährleistet [HBR09a].

3.6 Zusammenfassung

In diesem Kapitel werden die konventionellen Ansätze für das Management von Prozessvarianten erläutert und ihre jeweiligen Vor- und Nachteile aufgezeigt. Anschließend wird

die Grundidee von Provop vorgestellt. Diese basiert auf der Beobachtung, dass eine Prozessvariante üblicherweise aus einem bereits existierenden Prozessmodell durch variantenspezifische Anpassungen erstellt wird. Das Management von Prozessvarianten wird von Provop über den gesamten Prozesslebenszyklus betrachtet:

Bei der Modellierung der Prozessvarianten wird der Basisprozess sowie die variantenspezifischen Abweichungen mittels expliziten Änderungsoperationen erstellt. Sie beziehen sich auf konkrete Fragmente des Prozessmodells und mit ihnen können Prozessfragmente in den Basisprozess eingefügt, gelöscht und verschoben werden. Zudem können Attributwerte von Prozesselementen modifiziert werden.

Bei der Konfiguration der Prozessvarianten werden jene Optionen, d.h. Gruppen zusammengehörender Änderungsoperationen, ausgewählt und auf den Basisprozess angewandt, welche zur Bildung einer spezifischen Prozessvariante benötigt werden. Die Auswahl der Option basiert auf dem aktuellen Kontext der gewünschten Prozessvariante.

Bei der Ausführung von Prozessvarianten werden die konfigurierten Prozessmodelle in einem WfMS ausgeführt. In Provop wird darauf geachtet, dass die Atomarität von Optionen zur Laufzeit gewahrt wird sowie strukturelle und semantische Abhängigkeiten zwischen den Optionen eingehalten werden.

4 Verwandte Arbeiten

In diesem Kapitel werden verwandte Ansätze diskutiert, die sich mit dem Management von Prozessvarianten beschäftigen. Es werden aktuelle Forschungsansätze sowie konkrete Implementierungen betrachtet. Es wird ein Überblick über den aktuellen Stand der Technik gegeben. In Abschnitt 4.1 werden akademische Ansätze aus der Forschung diskutiert. Abschnitt 4.2 evaluiert heutige WfMS.

4.1 Ansätze aus der Forschung

Im Folgenden werden die wichtigsten Ansätze aus der Forschung vorgestellt. Zunächst wird in Abschnitt 4.1.1 das Konzept der Referenzprozessmodellierung erläutert und bestehende Ansätze diskutiert. Abschnitt 4.1.2 grenzt den Begriff der Flexibilität ein und diskutiert verschiedene Ansätze zur flexiblen Ausführung von Workflow-Modellen bzw. Prozessvarianten.

4.1.1 Referenzprozessmodellierung

Die Referenzprozessmodellierung stellt eine Erweiterung des Ein-Modell-Ansatzes dar (vgl. Abschnitt 3.1.2). Wie bereits erläutert, sind beim Ein-Modell-Ansatz die Informationen bzgl. der Variantenzugehörigkeit in der Ablauflogik versteckt. Der Modellierer kann nicht erkennen, ob eine bedingte Verzweigung eine Variante spezifiziert oder zum normalen Ausführungspfad gehört [HBR09a]. Um dieses Problem zu lösen, sind bei der Referenzprozessmodellierung für die Kennzeichnung der Prozessvarianten explizite Sprachelemente vorgesehen, wie z.B. Labels.

Mittels einer spezifischen Konfiguration für jeden Prozess wird aus dem Referenzprozessmodell die konkrete Prozessvariante abgeleitet. Das Referenzprozessmodell wird sozusagen prozess-spezifisch individualisiert [ADG+08, RA07].

In den folgenden Absätzen werden verschiedene Ansätze zur Referenzprozessmodellierung diskutiert. Dabei wird die jeweilige Idee und die Umsetzung beschrieben. Abschließend werden die Ansätze miteinander verglichen.

Konfigurierbare Knoten. Der Ansatz von Rosemann und van der Aalst [RA07] basiert auf *konfigurierbaren Knoten*. Die Autoren greifen das Problem auf, dass es in Referenzprozessmodellen keine Unterscheidung gibt zwischen herkömmlichen Strukturknoten, deren Pfadauswahl zur Laufzeit getroffen wird, und zwischen Strukturknoten, welche die Auswahl einer Variante darstellen. Für den Anwender scheint es so, als wären alle Knoten an Laufzeitentscheidungen gebunden. Die konfigurierbaren Teilbereiche eines Prozesses sind somit nur durch Analyse der Pfadauswahl erkennbar. Dementsprechend schlagen die Autoren eine Erweiterung einer existierenden Modellierungssprache vor, um die Variabilität in Prozessmodellen besser repräsentieren zu können. Als Ausgangssprache verwenden die Autoren Ereignis-Prozess-Ketten (EPK)[1] [KNS92], welche um konfigurierbare Knoten erweitert wird. Diese Knoten können so konfiguriert werden, dass sie Bestandteil des abgeleiteten Prozessmodells sind, bei der Ausführung ausgelassen werden oder optional sind. Um Abhängigkeiten zwischen den konfigurierbaren Knoten auszudrücken, können diese mit *Anforderungen* und *Richtlinien* verknüpft werden. Anhand der Wertebelegungen in den Anforderungen und Richtlinien wird das Referenzprozessmodell konfiguriert. Die erweiterten Sprachelemente können auf die Basiselemente von EPK abgebildet werden. Somit lässt sich der Ansatz in allen EPK-implementierenden Werkzeugen umsetzen.

Blocking und Hiding. Im Ansatz von Gottschalk et al. [GAJVR08] werden zwei Basisoperationen für die Konfiguration von Prozessmodellen eingeführt. Diese sind *blocking* (dt. blockieren, versperren) und *hiding* (dt. ausblenden, verbergen). Blockiert man eine Aktivität, so wird diese nicht ausgeführt und der Ausführungspfad, auf dem sie sich befindet, ausgelassen. Beim Ausblenden wird die Aktion geskippt (dt. ausgelassen), der Ausführungspfad wird aber nicht ausgelassen. Aktionen, die weder blockiert noch ausgeblendet werden, bezeichnet man als aktiviert. Die Autoren definieren für jede eingehende Kante an einer Aktion einen Eingangs-Port sowie für jede ausgehende Kante an einer Aktion einen Ausgangs-Port. Diese Ports können aktiviert oder blockiert werden. Mit Hilfe dieser

[1] Ereignis-Prozess-Ketten bestehen aus Funktionen, Ereignissen, logischen Konnektoren und Kanten.

konfigurierbaren Ports lässt sich nun aus einem gegebenen Referenzprozessmodell eine Prozessvariante ableiten: Im Referenzprozessmodell sind zunächst alle Ports aktiviert. Anschließend werden die Ports variantenspezifisch konfiguriert und im Ergebnismodell der Variante entsprechend Kanten und Aktivitäten zur Laufzeit ausgelassen.

Um eine dynamische Konfiguration zur Laufzeit zu ermöglichen, haben die Autoren zusätzlich optionales Blockieren und optionales Ausblenden eingeführt. Die Umsetzung des Ansatzes erfolgt in der Beschreibungssprache YAWL [AH05], welche für die Handhabung der konfigurierbaren Ports entsprechend erweitert wurde. Eine Transformation des erweiterten Modells in Standard-YAWL ist möglich.

Annotations-basierte Prozessvariabilität. Beim Ansatz von Puhlmann et al. [PSWW05] wird ein UML-Prozessmodell für die Darstellung der Variabilität um bestimmte Stereotypen[2] erweitert. An den entsprechenden Stellen im Prozessmodell, an denen variante Ausführungen möglich sein sollen, werden *Variation Points* eingefügt. Ein solcher Variation Point stellt eine abstrakte Aktivität dar, welche bei der Konfiguration durch eine konkrete variantenspezifische Aktivität ersetzt werden kann. Variantenspezifische Aktivitäten werden je einem Variation Point zugeordnet. Für die Variation Points gibt es diverse Annotierungen, mit denen man bspw. eine Default-Aktivität angeben oder Aktivitäten als optional kennzeichnen kann. Zudem darf die endgültige Auswahl der Aktivität erst dynamisch zur Laufzeit erfolgen. Die Umsetzung dieses Ansatzes ist mit UML Aktivitätendiagrammen [UML09] und mit BPMN [BPM09] möglich.

Aggregierte Ereignis-Prozess-Ketten. Der Ansatz von Reijers et al. [RMT09] basiert auf EPK [KNS92]. Hierbei geht es darum, aus einem aggregierten Referenzprozessmodell einzelne Varianten bzw. Familien von Varianten zu extrahieren. Dazu werden alle Funktionen und Ereignisse mit Beschriftungen, sog. Labels, angereichert. Diese Labels geben darüber Auskunft, zu welcher Variante das Prozesselement gehört. Somit ist es mit Hilfe der Labels möglich aus dem aggregierten Prozessmodell einzelne Prozessvarianten oder Familien von Prozessvarianten zu extrahieren. Um die Zugehörigkeit zu Familien von Varianten ausdrücken zu können, entsprechen die Labels einer baumartigen Hierarchie mit einer Wurzel. Die Blattknoten der Baumstruktur repräsentieren einzelne Prozessvarianten, die Knoten mit Kindern hingegen repräsentieren Familien von Prozessvarianten. Somit stellt jeder Vaterknoten eine Generalisierung und jeder Kindknoten eine Spezialisierung dar. Die

[2] Ein Stereotyp bezeichnet in der Unified Modelling Language (UML) eine Erweiterung eines vorhandenen Modellelements. Stereotypen erlauben dem Modellierer das UML-Vokabular mit Elementen zu erweitern, welche von existierenden Elementen abgeleitet werden.

extrahierte, mit Labels aggregierte Prozessvariante lässt sich in eine EPK transformieren, da die Labels nur Beschriftungen sind und keine neuen Ausführungssemantiken aufweisen.

Vergleich. Bei allen Ansätzen zeigt sich, dass im Referenzprozessmodell mehrere Varianten eines Prozesses abgebildet werden können. Zur Konfiguration des Referenzprozessmodells gibt es in jedem Ansatz spezielle Sprachelemente bspw. konfigurierbare Knoten. Allerdings muss bei allen Ansätzen das Referenzprozessmodell alle Prozesselemente der Prozessvarianten enthalten. Ein Einfügen von Prozessfragmenten während der Konfiguration ist nicht möglich. Somit müssen schon während des Erstellens alle möglichen Pfade bekannt sein. Des Weiteren ist ein Verschieben von Aktivitäten nicht möglich. Um dies darstellen zu können, muss die Aktivität mehrfach an unterschiedlichen Stellen in das Referenzprozessmodell aufnehmen. Dadurch werden die Referenzprozessmodelle komplex. Ein explizites Anpassen von Attributen ist in keinem Ansatz vorgesehen.

Damit während der Ausführung zwischen Varianten eines Prozesses gewechselt werden kann, müssen dynamische Anpassungen zur Laufzeit vorgenommen werden können. Eine optionale Konfiguration wie im Ansatz Gottschalk et al. erfüllt dieses Kriterium, da erst zur Laufzeit die Konfigurationsentscheidung getroffen wird und dementsprechend auch der Ausführungspfad erst zur Laufzeit bestimmt wird [GAJVR08]. Einzig der Ansatz der aggregierten EPK sieht keine optionale Konfigurationsmöglichkeit vor.

Keiner der vorgestellten Ansätze unterstützt das Ändern von Attributwerten von Aktivitäten. Das Verschieben von Prozessfragmenten ist nur durch Modellredundanzen abbildbar. Somit stellt keiner der Ansätze eine zufriedenstellende Lösung im Bereich des Managements von Prozessvarianten dar.

4.1.2 Flexibilität im Prozessmanagement

Unter Prozessflexibilität wird i.A. die Fähigkeit bezeichnet, mit vorhersehbaren und unvorhersehbaren Änderungen umzugehen und hierbei nur jene Teilbereiche eines Prozesses anzupassen, welche effektiv von den Änderungen betroffen sind. Die restlichen Teile des Prozesses bleiben unberührt. Zur Umsetzung der Prozessflexibilität existieren folgende vier Ansätze [SMR+08]:
Bei der *Flexibilität durch Design* werden alle bekannten Alternativen in das Prozessmodell integriert. Die Flexibilität wird durch die Auswahl des gewünschten Pfades bei der Ausführung erreicht. Die gewünschten Alternativen werden als bedingte Verzweigungen model-

liert. Ein Beispiel für diesen Ansatz bietet die Referenzprozessmodellierung [SMR+08].
Flexibilität durch Abweichung bedeutet zur Laufzeit vom gegebenen Ausführungspfad abzuweichen. Diese Abweichungen gelten nur auf der Ebene einzelner Prozessinstanzen, nicht für das Prozessmodell. Abweichungen werden durch Auslassen, Zurücksetzen und Wiederholen von einzelnen Aktivitäten erreicht [SMR+08].
Flexibilität durch Unterspezifizierung bedeutet bei der Modellierung noch nicht den gesamten Prozess exakt zu modellieren, sondern noch nicht feststehende Bereiche mit Platzhaltern aufzufüllen. Diese Platzhalter werden zur Laufzeit spezifiziert, d.h. durch Aktivitäten oder Sub-Prozesse ersetzt [SMR+08].
Flexibilität durch Änderung bedeutet zur Laufzeit ggf. alle Prozessinstanzen auf ein neues Prozessmodell zu migrieren. Hier muss für jede Prozessinstanz geprüft werden, ob sie auf das neue Prozessmodell migriert werden kann. Es muss genau festgelegt werden, welche Prozessinstanzen abgebrochen und neugestartet werden und welche Prozessinstanzen mit dem alten Prozessmodell beendet werden [WRRM08].

Provop realisiert die Ansätze der Flexibilität durch Design sowie der Flexibilität durch Unterspezifizierung. Auf Ansätze zur Erreichung der Flexibilität durch Design wurde bereits in Abschnitt 4.1.1 eingegangen. Die Flexibilität durch Unterspezifizierung wird in Provop durch die dynamische Konfiguration erreicht. Die Auswahl, welche Änderungsoperationen ausgeführt werden sollen und welche nicht, erfolgt erst zur Laufzeit. Im Folgenden werden verwandte Ansätze zur Erreichung dieser Art der Flexibilität diskutiert.

Late Binding. Beim Late Binding wird die Auswahl der konkreten Implementierung einer Aktivität zur Laufzeit aus einer Menge von vorher definierten Aktivitäten getroffen. Die Entscheidung dazu basiert auf Regeln oder Benutzerentscheidungen. Während der Modellierungszeit wird ein Platzhalter modelliert, der zur Laufzeit durch eine konkrete Aktivität ersetzt wird [WSR09].

Eine Umsetzung des Late Bindings stellt der Ansatz von Adams et al. [AHEA06] dar. In diesem Ansatz werden die Platzhalter als sog. *Worklets* definiert. Ein Worklet ist ein erweiterbarer Katalog an Aktivitäten, Sub-Prozessen und zugehörigen Regeln für die Auswahl der Aktivitäten oder Sub-Prozesse. Ein Worklet ist als Dienst (engl. Service) in der YAWL-Umgebung [AH05] implementiert. Beim Aufruf des Dienstes wird unter Einbeziehung von Regeln und des Kontextes dynamisch der zu bindende Dienst aus dem Katalog des Worklets gewählt. Die Dienste können weitere Dienste einbinden. Der Katalog ist zur Laufzeit erweiterbar.

Ähnlich wie bei Provop steht beim Late Binding das Grundgerüst des Prozessmodells während der Konfiguration fest. Zur Laufzeit werden die Platzhalter durch Prozessfragmente ersetzt. Diese sind vor der Ausführung bekannt. Wie bei Provop weiß der Anwender schon vor der Ausführung, welche Pfade und Aktivitäten im Prozessmodell ausführbar sind. Allerdings weiß er aufgrund der kontextabhängigen Entscheidungen zur Laufzeit nicht im Voraus welche tatsächlich ausgeführt werden.

Late Modeling. Das Late Modeling erlaubt das "späte Modellieren" von ganzen Prozessfragmenten zur Laufzeit. Die Fragmente basieren auf Aktivitäten und Bedingungen, welche zur Modellierungszeit erstellt werden. Die Zusammensetzung der Aktivitäten zu einem Prozessfragment erfolgt erst zur Laufzeit. Zur Modellierungszeit wird an Stelle des Prozessfragments ein Platzhalter erstellt [WSR09].

Ein Beispiel für das Late Modeling ist der Ansatz von Sadiq et al. [SOS05]. Bei diesem Ansatz besteht ein Prozess aus dem Kernprozess sowie aus sog. *Pockets of Flexibility*. Der Kernprozess stellt einen normalen Prozess dar, bestehend aus Aktivitäten und Kontrollfluss. Die Pockets werden im Kernprozess von Platzhaltern repräsentiert. Sie bestehen aus einer Menge von Prozessfragmenten und einer Menge von Bedingungen für deren Zusammensetzung. Die konkrete Implementierung der Pockets erfolgt erst zur Laufzeit. Dort können die Fragmente der Pockets unter Beachtung der Bedingungen beliebig kombiniert werden. Der Ansatz erlaubt somit eine volle Spezifikation des Prozessmodells zur Laufzeit.

Im Gegensatz zu Provop weiß der Anwender nach der Konfiguration beim Late Modeling noch nicht, wie das Prozessmodell aussieht. Die Zusammenstellung des Prozessmodells erfolgt erst zur Laufzeit.

Late Composition. Das Late Composition erlaubt das Zusammenstellen existierender Prozessfragmente zu einem Prozessmodell während der Ausführung. Es gibt kein vordefiniertes Schema. Die Prozessinstanz wird durch Auswahl verfügbarer Fragmente und unter Beachtung von vordefinierten Bedingungen ad-hoc erstellt [RRMD09].

Ein Beispiel für Late Composition ist *DECLARE* von Pesic et al. [PSSA07]. Ähnlich wie bei Sadiq et al. [SOS05] wird das Prozessmodell als eine Menge von Aktivitäten und Bedingungen definiert. Zur Laufzeit wird für jede Prozessinstanz ein individuelles Prozessmodell aus den zur Verfügung stehenden Aktivitäten unter Beachtung der Bedingungen zusammengestellt. Das gesamte Prozessmodell wird zur Laufzeit erstellt. Der Ansatz kann auch in Kombination mit imperativen Modellierungssprachen wie bspw. YAWL [AH05] verwendet werden.

Ähnlich wie beim Late Modeling kann beim Late Composition die Zusammensetzung der konkreten Bestandteile des Prozessmodells erst zur Laufzeit erfolgen. Dies ist in Provop nicht vorgesehen, aber für ein Variantenmanagement nicht erforderlich.

4.2 Ansätze aus der Praxis

In diesem Abschnitt werden wichtige WfMS untersucht und diskutiert. Die evaluierten Produkte sind ADEPT, Tibco iProcess Suite, WebSphere Integration Developer und das YAWL-System. Die untersuchten Aspekte sind die vom WfMS unterstützte Prozessbeschreibungssprache, explizite Konzepte für das Management von Prozessvarianten und die Flexibilität zur Laufzeit (vgl. Abschnitt 4.1.2).

4.2.1 ADEPT

ADEPT ist ein Forschungsprojekt des Instituts für Datenbanken und Informationssysteme an der Universität Ulm im Bereich Prozessmanagement-Systeme. Das Projekt wurde Mitte der 90er Jahre gestartet und war von 2004 bis 2007 Teil eines Forschungsprojekts des Landes Baden-Württemberg. Seit 2008 wird die ADEPT-Codebasis in der kommerziellen AristaFlow®BPM-Suite zur Produktreife geführt [DRRM+09].

ADEPT verwendet zur Modellierung und Ausführung von Prozessmodellen Well-Structured-Marking-Netze (WSM-Netze) (vgl. Reichert et al. [RRMD09]). In WSM-Netzen werden Kontrollstrukturen wie Sequenzen, Verzweigungen oder Schleifen auf Blöcke mit eindeutigen Start- und Endknoten abgebildet (vgl. Abbildung 4.1). Prozessinstanzen werden in WSM-Netzen anhand einer Markensituation beschrieben. Jede Aktivität besitzt einen Status und jede Kante eine Markierung. Für abgewählte Pfade gibt es in ADEPT eine Dead-Path-Elimination.

ADEPT bietet keine expliziten Konstrukte für das Variantenmanagement an. Zwar lassen sich Prozessinstanzen zur Laufzeit adaptieren, jedoch können solche Adaptionen nicht dauerhaft gespeichert und wiederverwendet werden. Dies ist allerdings durch den auf ADEPT aufbauenden Ansatz von Weber et al. [WWRD07] möglich. Es lassen sich mit ADEPT nur die konventionellen Ansätze für das Management von Prozessvarianten umsetzen (vgl. Abschnitt 3.1).

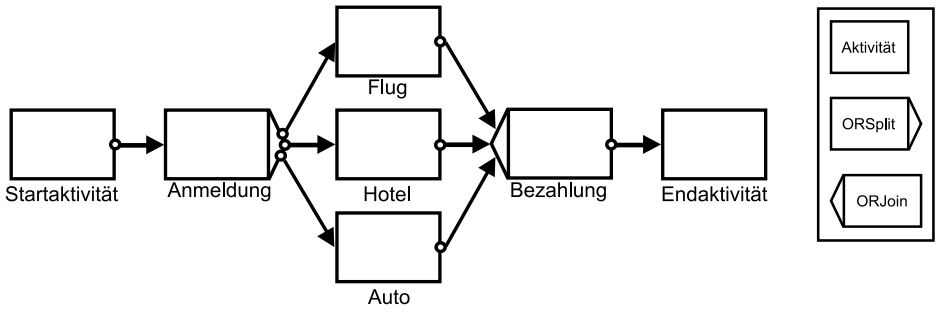

Abbildung 4.1: WSM-Netz in ADEPT

Der Begriff des Kontexts hängt bei ADEPT mit der Adaption von Prozessinstanzen zusammen. Anhand des Kontexts kann ausgewertet werden, welche Änderungen in der aktuellen Ausführungssituation möglich sind. Dem Benutzer werden nur die entsprechenden Änderungsoperationen angeboten. Dabei ist zu beachten, dass nur solche Änderungsoperationen zugelassen sind, die einen konsistenten Prozessgraphen wieder in einen neuen konsistenten Prozessgraphen überführen.

In ADEPT können einzelne Prozessinstanzen auf ein neues Prozessschema migriert werden. Des Weiteren ist auch eine Prozessschema-Evolution zur Laufzeit realisierbar. Dabei wird auf Basis des bisherigen Prozessschemas, des neuen Prozessschemas, der aktuellen Ausführungssituation der Prozessinstanz sowie ggf. vorgenommenen Ad-hoc-Änderungen ermittelt, ob eine Prozessinstanz auf das neue Prozessschema migriert werden kann.

4.2.2 Tibco iProcess Suite

Das WfMS Staffware wurde 2004 von Tibco übernommen und wird seitdem als Tibco® Staffware Process Suite weitervertrieben. Das System ist ein kommerzielles Produkt im Bereich des Prozessmanagements und besteht aus mehreren Anwendungsmodulen.

Zur Modellierung von Prozessen im Business Studio wird BPMN verwendet. Die Ausführung der Prozesse im XPDL-Format erfolgt in der iProcess Engine. Rahmenbedingungen und Geschäftsregeln können im Rules Manager definiert werden. Dadurch lassen sich Ausführungsregeln für die Prozesse erstellen [TIB05].

Tibco iProcess Suite bietet keine expliziten Konstrukte für das Management von Prozessvarianten. Jedoch lassen sich auch in Staffware die konventionellen Ansätze für das Management von Prozessvarianten umsetzen.

Die Tibco iProcess Suite lässt während der Ausführung Ad-hoc-Änderungen zu. Es können einzelne Aktivitäten geskippt, neu gestartet oder auch komplett ausgelassen werden. Zudem ist ein Late Binding auf Type-Level möglich. Dies bedeutet, dass für einzelne Prozessinstanzen die konkrete Implementierung einiger Aktivitäten erst zur Laufzeit festgelegt werden kann [WRRM08].

4.2.3 WebSphere Integration Developer

Der WebSphere Integration Developer ist Teil der WebSphere-Produktfamilie von IBM. Dort findet die Modellierung der ausführbaren Prozesse und die Abbildung der Aufrufstruktur der Prozesse statt. Die Prozesse werden im WebSphere Process Server ausgeführt [IBM09].

Die Modellierung und Ausführung wird BPEL verwendet. Obwohl BPEL eine textuelle Prozessbeschreibungssprache ist, gibt es im WebSphere Integration Developer einen graphischen BPEL-Editor zur einfacheren Modellierung der Prozesse. Der BPEL-Standard wird um einige proprietäre Konstrukte erweitert, wie bspw. den Java Snippet Activities.

Der WebSphere Integration Developer bietet keine expliziten Konstrukte für das Variantenmanagement. Die konventionellen Ansätze sind realisierbar.

Rahmenbedingungen für die Ausführung lassen sich mittels Business Rules abbilden. Diese spezifizieren Regeln für die Ausführung der Prozessinstanzen. Abhängig vom Ergebnis einer Business Rule wird eine Aktion ausgelöst. Die Regellogik kann auch zur Laufzeit modifiziert werden.

Bei der Ausführung können Aktivitäten einzelner Prozessinstanzen übersprungen und somit ausgelassen werden. Rücksprünge sind ebenfalls möglich. Allerdings wird hier ein korrektes Schreiben von Datenobjekten nicht mehr gewährleistet. Zudem können zur Laufzeit Prozessschritte als Sub-Aktivitäten anderer Aktivitäten eingefügt werden. Eine Änderung der Prozessinstanz zur Laufzeit ist nicht möglich. Konzepte zur Evolution von Prozessvarianten sind nicht realisiert.

4.2.4 YAWL-System

Das YAWL-System ist ein WfMS auf Basis der in Abschnitt 2.2.4 vorgestellten Prozessbeschreibungssprache YAWL. Das System ist modular aufgebaut und besteht aus mehreren Komponenten.

Die Modellierung von Prozessen erfolgt im YAWL-Designer durch die Beschreibungssprache YAWL. Die bestehenden Spezifikationen der Prozesse werden anschließend im YAWL-Repository abgelegt und schließlich in der YAWL-Engine instanziiert und ausgeführt.

Für das explizite Management von Prozessvarianten bietet YAWL den in Abschnitt 4.1.1 beschriebenen Ansatz von Gottschalk et al. [GAJVR08] an. Dieser Ansatz der Referenzprozessmodellierung wird durch die Konfiguration der Knoten im Prozessmodell realisiert. Des Weiteren lässt sich in YAWL auch die Technik des Mehr-Modell-Ansatzes problemlos umsetzen.

In YAWL sind die in Abschnitt 4.1.2 vorgestellten Ansätze des Late Bindings von Adams et al. [AHEA06] sowie der Late Composition von Pesic et al. [PSSA07] in YAWL umsetzbar. Des Weiteren ist durch den Ansatz von Adams et al. [AHAE07] Flexibilität durch Unterspezifizierung möglich. Somit bietet YAWL gute Möglichkeiten zur Gewährleistung der Flexibilität während der Ausführung von Prozessinstanzen.

4.2.5 Vergleich der Workflow-Management-Systeme

Alle vorgestellten WfMS verwenden zur Modellierung und Ausführung von Prozessmodellen mächtige Beschreibungssprachen. Während die ADEPT und YAWL wissenschaftlich entwickelte Beschreibungssprachen einsetzen, verwenden die WfMS von Tibco und IBM standardisierte Sprachen wie bspw. BPEL.

Von den betrachteten WfMS bietet nur das YAWL-System explizite Konstrukte für das Management von Prozessvarianten an. Die restlichen WfMS sehen hierzu keine expliziten Konstrukte vor. Allerdings lassen sich die in Abschnitt 3.1 diskutierten konventionellen Ansätze in allen Systemen abbilden.

Im Bereich der Flexibilität zur Laufzeit unterscheiden sich die angebotenen Funktionalitäten der einzelnen Systeme. ADEPT bietet mit der Prozessschema-Evolution ein sehr mächtiges Konzept an. YAWL stellt ebenfalls Konzepte für die Ermöglichung der Flexibilität während der Ausführung zur Verfügung. Tibco iProcess Suite und WebSphere Integration Developer beschränken sich auf Ad-hoc-Änderungen zur Laufzeit.

Eine Betrachtung ob und in welcher Form sich der in Kapitel 3 vorgestellte Lösungsansatz von Provop zum Management von Prozessvarianten in den vorgestellten Systemen umsetzen lässt, erfolgt in Kapitel 6.

4.3 Zusammenfassung

In diesem Kapitel werden sowohl wissenschaftliche als auch kommerzielle Ansätze zur Thematik von Prozessvarianten betrachtet. Zunächst wird der Ansatz der Referenzprozessmodellierung diskutiert und der Begriff der Flexibilität im Prozessmanagement erläutert. Es werden die verschiedenen Systematiken vorgestellt und einige konkrete Ansätze und Implementierungen beschrieben.

Im zweiten Teil des Kapitels werden verschiedene WfMS vorgestellt. Zu jedem System wird evaluiert, welche Beschreibungssprache verwendet wird, ob es explizite Konstrukte für das Management von Prozessvarianten gibt, ob man den Kontext von Varianten abbilden kann und welche Techniken zur Prozessflexibilität während der Ausführung vorhanden sind.

5 Anforderungen an die Ausführung von Prozessvarianten in Provop

In diesem Kapitel wird diskutiert, welche Anforderungen die Provop-Konzepte an die Ausführung von technischen Workflow-Modellen stellen. Abschnitt 5.1 erläutert die Anforderungen, die sich durch die Verwaltung eines domänenspezifischen Kontexts ergeben. In Abschnitt 5.2 werden die Anforderungen dargestellt, welche sich bei der Gewährleistung der Optionsbeziehungen zur Laufzeit ergeben. Abschnitt 5.3 erklärt, welche Auswirkungen die Provop-spezifischen Kontrollfluss-Konstrukte für die Ausführung von Prozessvarianten haben und erörtert daraus resultierende Anforderungen. Abschnitt 5.4 gibt einen Überblick über die Anforderungen und fasst das Kapitel zusammen.

5.1 Anforderungen aus der Kontextabhängigkeit von Optionen

Wie in Abschnitt 3.4 erläutert, hängt die Auswahl von Prozessvarianten bzw. von Optionen für die Konfiguration der Prozessvarianten vom domänenspezifischen Kontext des Prozesstyps ab. Dieser wird dazu in einem Kontextmodell durch Kontextvariablen beschrieben. Die Relevanz von Optionen in einem gegebenen Kontext wird über Kontextbedingungen beschrieben. Bei der Ausführung einer Instanz einer Prozessvariante muss hierzu ein Kontextmodell mit den Kontextinformationen verwaltet werden. Daraus ergeben sich folgende Anforderungen:

Anforderung 1: Verwaltung der Kontextinformationen. Die Informationen des Kontexts müssen in Form von Kontextvariablen in einem Kontextmodell verwaltet werden können. Bei der Verwaltung der Daten der Kontextvariablen muss darauf geachtet werden, dass die Daten nicht redundant und im korrekten Format gespeichert sind. Die Aktivitäten der

Prozessinstanz müssen zu jeder Zeit auf die Daten des Kontextmodells zugreifen können. Es muss sowohl ein lesender als auch ein schreibender Zugriff möglich sein, bspw. wenn Aktivitäten Kontextinformationen als Ausgabedaten erzeugen.

Anforderung 2: Aktualisierung von Kontextvariablen. Für jede Kontextvariable muss ein *Modus* spezifiziert werden können, der die Werte *statisch* oder *dynamisch* annehmen kann. Änderungen an Kontextvariablen mit Modus dynamisch müssen zur Laufzeit erkannt und verarbeitet werden können. Änderungen zur Laufzeit an Kontextvariablen mit Modus statisch dürfen hingegen keine Auswirkungen auf die Ausführung haben. Ihr Wert muss bei Prozessstart vorliegen.

Anforderung 3: Geltungsbereich der Kontextinformationen. Bei der Ausführung muss zwischen verschiedenen Arten von Kontextinformationen unterschieden werden: Es gibt Daten, die nur für eine spezifische Prozessinstanz gelten und nur für diese Prozessinstanz relevant sind. Des Weiteren gibt es Daten, die für alle Prozessinstanzen eines Prozesstyps gelten und somit u.U. für mehrere Prozessinstanzen relevant sind. Dementsprechend muss bei der Verwaltung der Kontextinformationen zwischen den verschiedenen Geltungsbereichen unterschieden werden.

5.2 Anforderungen durch Optionsbeziehungen

In Provop können strukturelle und semantische Abhängigkeiten, sog. Optionsbeziehungen, zwischen den Optionen definiert werden (vgl. Abschnitt 3.4.2). Diese Beziehungen sind aufgrund der dynamischen Anwendung von Optionen auch während der Ausführung relevant. Hier ergeben sich folgende Anforderungen:

Anforderung 4: Verwaltung der Optionsbeziehungen. Abhängigkeiten zwischen Optionen müssen zur Laufzeit verwaltet werden, um die Kompatibilität dynamisch angewandter Optionen mit den bisher angewandten Optionen zu gewährleisten. Für jede Abhängigkeit bedarf es einer korrekten und vollständigen Angabe der Optionsbeziehung. So müssen bspw. bei der *Auswahl n-aus-m* ein Vergleichsoperator, die Zahl *n* sowie die Menge der *m* Optionen spezifiziert werden.

Anforderung 5: Auswertung der Optionsbeziehungen. Die Prozessinstanz eines Variantenmodells muss Zugriff auf die verwalteten Optionsbeziehungen haben, damit während

der Ausführung einer Prozessinstanz die zulässigen Kombinationen für die Anwendung der Optionen ermittelt werden können.

5.3 Anforderungen durch Provop-spezifische Kontrollfluss-Konstrukte

Zur korrekten Modellierung und Konfiguration von Prozessvarianten wurden in Provop verschiedene Kontrollfluss-Konstrukte und Elemente eingeführt: Zur Referenzierung von zu ändernden Prozessfragmenten durch Änderungsoperationen wurden sog. Aufsetzpunkte eingeführt. Diese Aufsetzpunkte sind für die Ausführung von Prozessvarianten nicht relevant und werden daher aus dem Ergebnismodell entfernt. Somit spielen diese Konstrukte bei der Ausführung der Prozessvarianten keine Rolle mehr.

Durch die in Abschnitt 3.3.2 eingeführten höherwertigen Änderungsoperationen werden die Prozessmodelle modifiziert. Es können Prozessfragmente eingefügt, gelöscht und verschoben werden. Zudem können Attributwerte von Aktivitäten geändert werden. Diese Änderungsoperationen können Bestandteil von statischen oder dynamischen Optionen sein. Damit diese neuen Prozessfragmente durch das Einfügen korrekt in das Prozessmodell des Basisprozesses integriert werden können, bedarf es einer entsprechenden Aufsplittung und Synchronisation des Kontrollflusses im Prozessmodell.

Um eine korrekte Aufsplittung und Synchronisation zu erreichen, werden die Prozessfragmente der Änderungsoperationen mit zwei neuen Typen von Strukturknoten in das Prozessmodell integriert. Diese neuen Strukturknoten sind der *OPSplit-Knoten* für die Aufsplittung und der *OPJoin-Knoten* für die Synchronisation der Prozessfragmente. Mittels der neuen Konstrukte können die einzelnen Änderungsoperationen gekapselt werden. Die neu integrierten Prozessfragmente werden über sog. *Operationskanten* mit den Strukturknoten verbunden. Beispiel 5.1 beschreibt die Auswirkungen auf das Basismodell durch die Integration der neuen Strukturknoten.

Beispiel 5.1 (Dynamisches Verschieben) Abbildung 5.1 zeigt die Auswirkungen auf das Basismodell durch ein dynamisches Verschieben eines Prozessfragments. Das Verschieben setzt sich im Prinzip aus einer Löschen- und Einfüge-Operation zusammen (vgl. Abschnitt 3.3.2). Im Basisprozess (a) werden die Prozessschritte A und B sequentiell ausgeführt. Die Option sieht ein Verschieben von A parallel zu B vor (b). Die Kontextbedingung

B1 der Option bezieht sich auf eine dynamische Kontextvariable, Option 1 wird daher dynamisch auf den Basisprozess angewandt: Entsprechend des dynamischen Verschiebens wird Aktivität A von der ursprünglichen Position im Prozessmodell entfernt und an der neuen Position wieder eingefügt, falls die Kontextbedingung B1 der Option erfüllt ist. Ist die Kontextbedingung B1 nicht erfüllt, so bleibt Aktivität an der ursprünglichen Position. Zur korrekten Aufsplittung und Synchronisation des Kontrollflusses werden im Ergebnismodell OPSplit- und OPJoin-Knoten eingefügt (c). An den OPSplit-Knoten erfolgt die Überprüfung, ob die Änderungsoperationen der Option angewandt werden sollen.

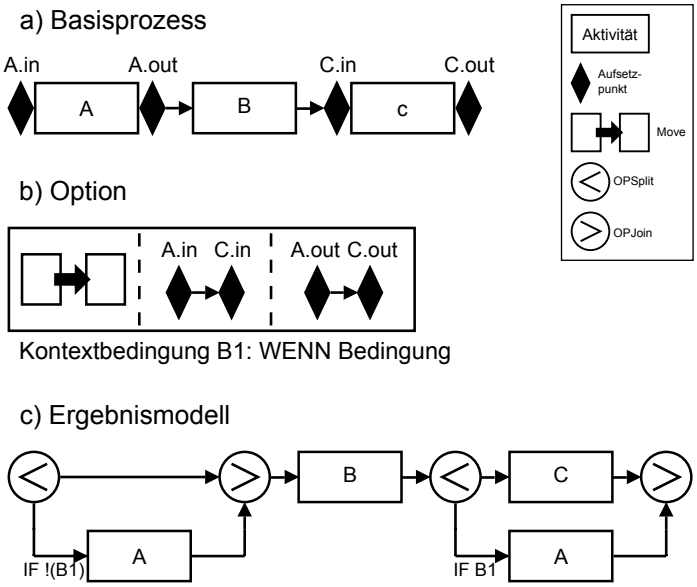

Abbildung 5.1: Dynamisches Verschieben eines Prozessfragments

Durch die Aufsplittung und Synchronisation des Kontrollflusses während der Ausführung ergeben sich folgende Anforderungen:

Anforderung 6: Prüfung der Anwendbarkeit einer dynamischen Option. Bei der Aufsplittung muss über die Anwendung einer Änderungsoperation einer Option entschieden werden. Handelt es sich um den ersten zur Laufzeit erreichten OPSplit-Knoten einer Option, wird geprüft, ob die Kontextbedingung der Option erfüllt ist. Ist dies der Fall, so wird geprüft, ob die Anwendung der Option kompatibel mit allen Optionsbeziehungen ist. Ist dies der Fall, so wird die Option angewandt. Ist eine Beziehung zwischen den bisher angewandten Optionen verletzt, so darf aufgrund der höheren Priorisierung harter Optionsbeziehungen die Option trotz gültiger Kontextbedingung nicht angewandt werden. Ist die Kontextbe-

dingung der Option nicht erfüllt, so muss dennoch geprüft werden, ob die Option aufgrund der Abhängigkeiten zwischen den Optionen angewandt werden muss. Abbildung 5.2 zeigt das Vorgehen zur Prüfung der Anwendbarkeit.

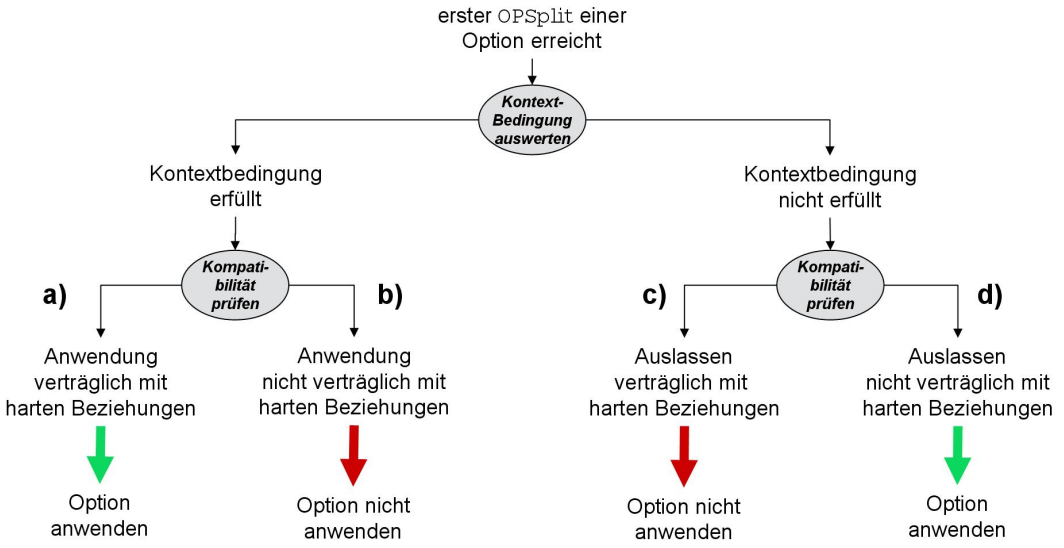

Abbildung 5.2: Prüfung der Anwendbarkeit einer dynamischen Option

Anforderung 7: Gewährleistung der Atomarität von Optionen. Zur Gewährleistung der Atomarität von Optionen (siehe Abschnitt 3.5) muss an allen OPSplit-Knoten einer Option die gleiche Entscheidung getroffen werden. Wurde bspw. positiv über die Anwendung einer Änderungsoperation einer dynamischen Option entschieden, so müssen auch alle weiteren Änderungsoperationen dieser Option angewandt werden. Bei einer negativen Entscheidung darf gar keine Änderungsoperation dieser Option angewandt werden.

Anforderung 8: Gewährleistung der korrekten Synchronisation. Bei der Synchronisation der Prozessfragmente muss darauf geachtet werden, dass die wieder korrekt zusammengeführt werden. In Provop ist es nicht erlaubt, dass bereits abgewählte Pfade durch die Integration von Prozessfragmenten aus Änderungsoperationen wieder aktiviert werden können. Der OPJoin-Knoten muss sicherstellen, dass dies nicht vorkommen kann. Beispiel 5.2 stellt diese Problematik dar.

Beispiel 5.2 (Synchronisation am OPJoin-Knoten) Abbildung 5.3 zeigt einen Anwendungsfall zur Gewährleistung der korrekten Synchronisation am OPJoin-Knoten. Aktivität E wurde mittels einer Änderungsoperation in das Prozessmodell eingefügt. Während der

Ausführung des gezeigten Prozessmodells wird der untere Pfad mit Aktivität C gewählt und aufgrund des XORSplits der obere Pfad mit Aktivität A abgewählt. Am OPSplit-Knoten wird entschieden, den eingefügten Pfad ebenfalls auszuführen. Dadurch wird nach Aktivität C auch Aktivität E ausgeführt. Am OPJoin-Knoten ergibt sich dadurch folgende Situation: Kante x ist aktiviert, Kante y hingegen ist abgewählt, da der gesamte obere Pfad abgewählt wurde. Bei der Synchronisation am OPJoin-Knoten muss sichergestellt werden, dass der obere Pfad mit Aktivität B nicht aktiviert wird, da dieser Pfad am initialen XORSplit abgewählt wurde.

Durch die Synchronisation wird bspw. verhindert, dass es bei Aktivität B zu einer inkorrekten Datenversorgung kommt. Schreibt Aktivität A Daten, die bei Aktivität B gelesen werden, so werden die Daten bei der obigen Ausführungsreihenfolge nicht versorgt und in Aktivität B kommt es u.U. zu einem fehlerhaften Lesezugriff. Solche Szenarien dürfen in Provop nicht auftreten.

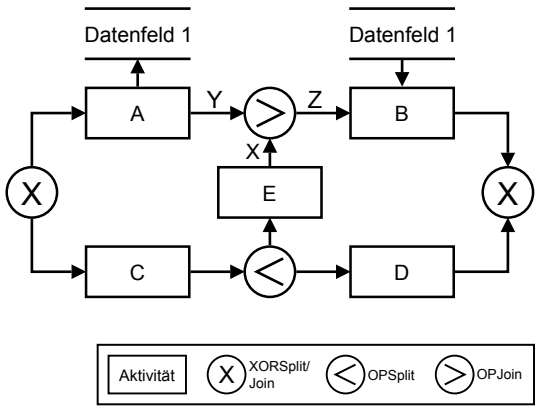

Abbildung 5.3: Anwendungsfall zur korrekten Synchronisation am OPJoin-Knoten

5.4 Zusammenfassung

In diesem Kapitel werden die Anforderungen an die Ausführung von Prozessvarianten in Provop vorgestellt. Abhängig vom Kontext, den strukturellen und semantischen Abhängigkeiten der Optionen sowie den Provop-spezifischen Kontrollfluss-Konstrukten ergeben sich unterschiedliche Anforderungen für die Überführung der Prozessmodelle in ausführbare Workflow-Modelle.

Zur Auswahl von Prozessvarianten und für die Konfiguration von Prozessvarianten müssen Kontextinformationen in einem Kontextmodell verwaltet werden können. Zusätzlich muss dabei zwischen statischen und dynamischen Kontextinformationen sowie nach verschiedenen Geltungsbereichen unterschieden werden können.

Zur Einhaltung der strukturellen und semantischen Abhängigkeiten zwischen den Optionen müssen Optionsbeziehungen verwaltet und während der Ausführung der Prozessinstanzen ausgewertet werden können.

Damit die Prozessfragmente aus den Änderungsoperationen korrekt in das Prozessmodell integriert werden können, bedarf es einer entsprechenden Aufsplittung und Synchronisation des Kontrollflusses. Bei der Aufsplittung muss über die Anwendbarkeit einer dynamischen Option entschieden und die Atomarität der Optionen gewährleistet werden können. Bei der Synchronisation muss ein korrektes Zusammenführen der Pfade gewährleistet werden können. In Tabelle 5.1 sind die Anforderungen nochmals zusammenfassend aufgelistet.

Zur Abbildung der Provop-Konzepte in einem technischen Workflow-Modell müssen Prozessbeschreibungssprachen bzw. WfMS die Anforderungen erfüllen. Im Folgenden wird daher vorgestellt, welche Lösungen in der Praxis und aktuellen Forschung existieren.

Tabelle 5.1: Anforderungen an die Ausführung von Prozessvarianten

Anforderung 1: Verwaltung der Kontextinformationen. Die Kontextinformationen müssen in einem Kontextmodell verwaltet werden.
Anforderung 2: Aktualisierung von Kontextvariablen. Es muss zwischen statischen und dynamischen Kontextvariablen unterschieden werden. Nur auf Änderungen an dynamischen Kontextvariablen darf zur Laufzeit reagiert werden.
Anforderung 3: Geltungsbereich der Kontextinformationen. Es muss zwischen Kontextinformationen, die nur für eine Prozessinstanz gelten, und Kontextinformationen, die für mehrere Prozessinstanzen gelten, unterschieden werden können.
Anforderung 4: Verwaltung der Optionsbeziehungen. Die strukturellen und semantischen Abhängigkeiten zwischen den Optionen, d.h. die Optionsbeziehungen, müssen verwaltet werden.
Anforderung 5: Auswertung der Optionsbeziehungen. Die Abhängigkeiten zwischen den Optionen müssen während der Ausführung ausgewertet und die zulässigen Kombinationen der Optionen ermittelt werden können.
Anforderung 6: Prüfung der Anwendbarkeit einer dynamischen Option. Vor der Anwendung dynamischer Änderungsoperationen muss geprüft werden, ob dies hinsichtlich des Kontextes und der harten Optionsbeziehungen zulässig ist.
Anforderung 7: Gewährleistung der Atomarität von Optionen. Während der Ausführung müssen entweder alle oder gar keine Änderungsoperationen einer dynamischen Option angewandt werden.
Anforderung 8: Gewährleistung der korrekten Synchronisation. Die Aktivierung von bereits abgewählten Pfaden während der Ausführung muss ausgeschlossen werden können, um dadurch fehlerhafte Ausführungsreihenfolgen und Prozessabbrüche zur Laufzeit zu verhindern.

6 Abbildung der Provop-Konzepte

In Kapitel 5 wurden die Anforderungen an die Ausführung vorgestellt, die sich durch die Provop-Konzepte ergeben. In diesem Kapitel werden geeignete Lösungsansätze für die Abbildung der Provop-Konzepte diskutiert. Dazu werden zu jeder Anforderungen alternative Abbildungsmöglichkeiten vorgestellt und jeweilige Vor- und Nachteile diskutiert. Des Weiteren wird evaluiert, wie die Lösungsansätze mit Prozessbeschreibungssprachen bzw. heutigen WfMS umgesetzt werden können.

Abschnitt 6.1 geht zunächst auf die Abbildung der Kontextabhängigkeit ein. Abschnitt 6.2 erörtert eine geeignete Abbildung für die strukturellen und semantischen Abhängigkeiten zwischen Optionen während sich Abschnitt 6.3 mit der Abbildung der Provop-spezifischen Kontrollfluss-Konstrukte befasst. In Abschnitt 6.4 wird diskutiert, wie die Abbildungsmöglichkeiten aus den vorigen Abschnitten in den Prozessbeschreibungssprachen bzw. WfMS umsetzbar sind.

6.1 Abbildung der Kontextabhängigkeit von Optionen

Nachdem in Abschnitt 5.1 die spezifischen Anforderungen der Abbildung der Kontextabhängigkeit erläutert wurden, wird in diesem Abschnitt auf verschiedene Abbildungssmöglichkeiten eingegangen.

6.1.1 Lösungsansätze

Anforderung 1 fordert, dass die Kontextinformationen in einem Kontextmodell verwaltet werden. Dazu sind verschiedene Ansätze denkbar:

- **Verwaltung als Prozessdaten.** Bei dieser Form der Datenverwaltung werden die Kontextinformationen als interne Prozessdaten repräsentiert. Im WfMS müssen ge-

eignete Datenstrukturen definiert werden, die die Kontextinformationen speichern können. Für jede Prozessinstanz wird ein Objekt definiert, welches alle notwendigen Kontextvariablen verwaltet. Jede Kontextvariable ist selbst ein Objekt, welches den Namen der Kontextvariable und den aktuellen Wert speichern kann.

Abbildung 6.1 skizziert die Verwaltung der Kontextinformationen als Prozessdaten. Die Daten sind direkt innerhalb der Prozessinstanz verfügbar, die Aktivitäten können auf Prozessdaten direkt zugreifen.

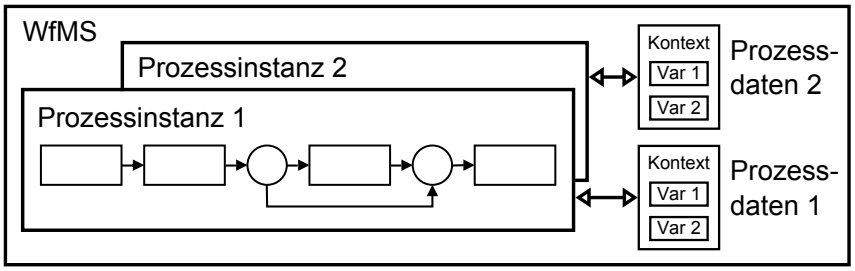

Abbildung 6.1: Verwaltung der Kontextinformationen als Prozessdaten

- **Verwaltung in einer Datenbank.** Hier werden die Kontextinformationen in einer Datenbank verwaltet. In der Datenbank muss ein geeignetes Speicherformat für die Daten erzeugt werden. In einer relationalen Datenbank bspw. muss eine Tabelle für die Kontextvariablen mit den nötigen Attributen für den Namen sowie den Wert generiert werden.

  Abbildung 6.2 veranschaulicht die Verwaltung der Kontextinformationen in einer Datenbank. Der Zugriff auf die Datenbank durch die Prozessinstanzen, welche im WfMS ausgeführt werden, erfolgt über entsprechende Schnittstellen des Systems. Der Aufbau der Verbindung zur Datenbank und die Zugriffe auf die Kontextinformationen durch Aktivitäten müssen zusätzlich programmiert werden.

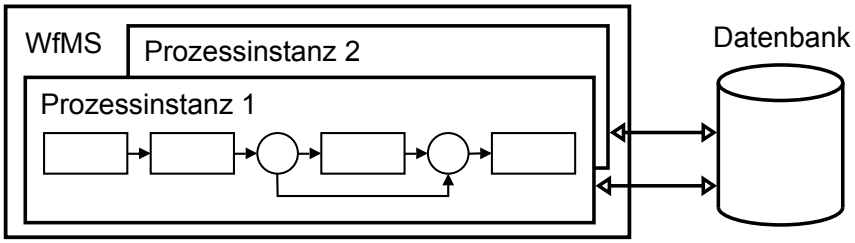

Abbildung 6.2: Verwaltung der Kontextinformationen in einer Datenbank

- **Verwaltung in externen Dateien.** Bei diesem Ansatz werden die Kontextinformationen in einer externen Datei gespeichert, wie z.B. einer Textdatei mit definiertem Format von Kontextvariablen, wie bspw. das CSV-Format [Sha05] oder das Speichern in einer XML-Datei [W3C08] nach einem definierten Schema.

  Abbildung 6.3 skizziert die Verwaltung der Kontextinformationen in einer externen Datei. Der lesende und schreibende Zugriff durch Aktivitäten auf die Datei muss entsprechend programmiert werden. Beim Schreiben der Datei muss darauf geachtet werden, dass dies im korrekten Format geschieht, damit ein fehlerfreier Zugriff gewährleistet werden kann.

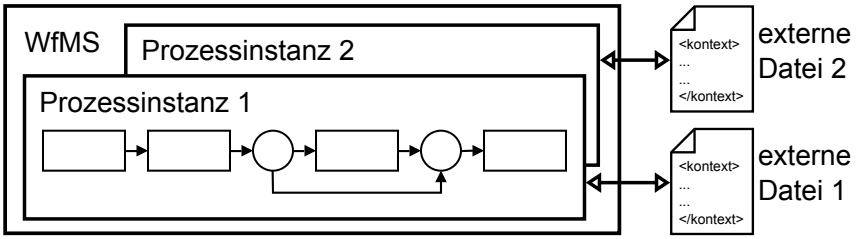

Abbildung 6.3: Verwaltung der Kontextinformationen in einer externen Datei

Gemäß **Anforderung 2** muss in einem Kontextmodell zwischen statischen und dynamischen Kontextvariablen unterschieden werden. Zu diesem Zweck kann für jede Kontextvariable ein zusätzliches Attribut spezifiziert werden, welches den Modus verwaltet. Bei der Speicherung in einer Datenbank wird dazu eine zusätzliche Spalte definiert. Zulässige Wertebelegungen für den Modus sind hierbei *statisch* und *dynamisch*.

Wie in Abschnitt 3.4.3 beschrieben, werden die Werte für statische Kontextvariablen einmal während der Konfiguration der Prozessvarianten festgelegt und dürfen sich anschließend nicht mehr ändern. Die Werte von dynamischen Kontextvariablen können sich hingegen jederzeit ändern. Dementsprechend dürfen Änderungen an statischen Kontextvariablen für die Ausführung keine Rolle mehr spielen. Statische Kontextvariablen müssen zur Laufzeit gesperrt werden. Änderungen an dynamischen Kontextvariablen während der Ausführung müssen erkannt und entsprechend verarbeitet werden.

Anforderung 3 besagt, dass für die Daten des Kontextmodells zwischen unterschiedlichen Geltungsbereichen der Kontextinformationen unterschieden werden muss. Beispiel 6.1 erläutert die Problematik für den Werkstattprozess.

Beispiel 6.1 (Unterschiedliche Geltungsbereiche der Kontextinformationen) Der `Kostenvoranschlag` für eine Reparatur eines Fahrzeugs wird bei der Diagnose des Fahrzeugs erstellt und ist für jedes Fahrzeug individuell. Dementsprechend gilt diese Kontextvariable nur jeweils für eine Prozessinstanz und muss für jede Prozessinstanz einzeln gepflegt werden.

Die `Auslastung` einer Werkstatt hängt davon ab, wie viele Aufträge vorhanden sind (die Anzahl der Aufträge ist gleichbedeutend mit den laufenden Prozessinstanzen einer bestimmten Werkstatt) und welche Kapazitäten der Werkstatt zur Verfügung stehen. Dementsprechend gilt der aktuelle Wert der Kontextvariable Auslastung für alle laufenden Prozessinstanzen des Werkstattprozesses. Diese Kontextvariable muss daher zentral für alle Prozessinstanzen gepflegt werden.

Während der Ausführung in Provop muss demnach zwischen Kontextinformationen, die nur für eine spezifische Prozessinstanz gelten, und Kontextinformationen, die für mehrere Prozessinstanzen eines oder mehrerer Prozesstypen gelten, unterschieden werden. In Provop werden diese beiden Arten im Folgenden als lokale und globale Kontextinformationen bezeichnet.

Um verschiedene Geltungsbereiche von Daten während der Ausführung von Prozessen in einem WfMS angeben zu können, unterscheidet Russell et al. [RHAE04] zwischen den folgenden Sichtbarkeiten der Daten:

- **Aktivitäten-spezifische Daten.** Die Datenelemente gelten nur für eine spezielle Aktivität und können nur von dieser Aktivität bearbeitet werden.
- **Block-spezifische Daten.** Unter einem Block versteht Russell einen ausgeführten Sub-Prozess einer Aktivität. Demnach gelten die Datenelemente für alle Komponenten des Sub-Prozesses.
- **Bereichs-spezifische Daten.** Die Datenelemente werden für eine Teilmenge aller Aktivitäten des Prozesses definiert und können von dieser Teilmenge bearbeitet werden.
- **Mehrfach-Instanz-spezifische Daten.** Diese Form der Sichtbarkeit gilt laut Russell für Aktivitäten, die während der Ausführung einer Prozessinstanz mehrfach instanziiert werden können. Die Datenelemente sind dann spezifisch für jede einzelne ausgeführte Instanz der Aktivität gültig.

- **Prozessinstanz-spezifische Daten.** Die Datenelemente werden für eine gesamte Prozessinstanz spezifiziert. Sie sind für alle Elemente der Prozessinstanz gültig und können von allen Elementen bearbeitet werden.
- **Prozesstyp-spezifische Daten.** Die Datenelemente werden für alle Prozessinstanzen eines Prozesstyps spezifiziert. Somit sind sie für alle Elemente aller Prozessinstanzen des Prozesstyps gültig.
- **WfMS-spezifische Daten.** Diese Datenelemente gelten für das gesamte WfMS und können von allen Elementen des WfMS bearbeitet werden. Insbesondere können Elemente von unterschiedlichen Prozesstypen die Datenelemente bearbeiten.

Die Unterscheidung lokaler und globaler Kontextvariablen ist mit Hilfe der von Russell beschriebenen Prozessinstanz-spezifischen sowie WfMS-spezifischen Daten abbildbar. Abbildung 6.4 stellt die Geltungsbereiche der lokalen (a) und globalen (b) Kontextvariablen dar.

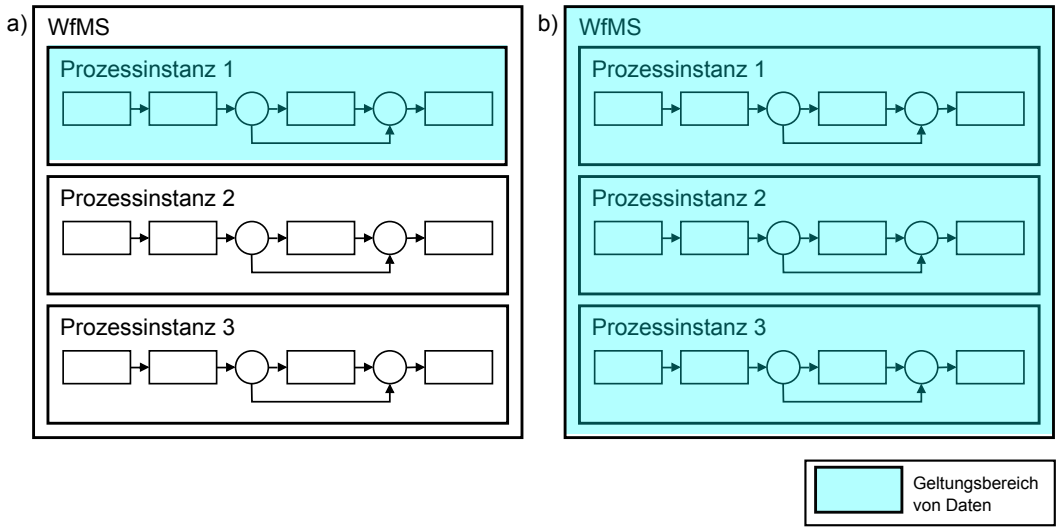

Abbildung 6.4: Geltungsbereiche a) lokaler und b) globaler Daten

Nachdem für die Kontextvariablen verschiedene Geltungsbereiche definiert wurden, müssen diese unterschiedlichen Arten von Kontextinformationen auch entsprechend ihrem Geltungsbereich verwaltet werden. Hierbei sind folgende Möglichkeiten denkbar:

- **Gemeinsame Verwaltung der lokalen und globalen Kontextinformationen.** Jede Prozessinstanz verwaltet sowohl die lokalen als auch die globalen Kontextinformatio-

nen in derselben Datenquelle. Bei Änderungen an den globalen Kontextinformationen müssen die veränderten Werte an alle betroffenen Prozessinstanzen dieses Prozesstyps weitergegeben werden.

- **Getrennte Verwaltung der lokalen und globalen Kontextinformationen.** Die lokalen und globalen Kontextinformationen werden getrennt voneinander verwaltet. Für jede Prozessinstanz gibt es eine spezifische Datenquelle mit den lokalen Kontextinformationen und für alle Prozessinstanzen eines Prozesstyps gibt es eine zentrale Datenquelle mit allen globalen Kontextinformationen.

6.1.2 Diskussion der Abbildungsmöglichkeiten

In diesem Abschnitt werden die oben genannten Lösungsansätze diskutiert. Um eine geeignete Abbildung für alle Anforderungen der Kontextabhängigkeit zu erhalten, müssen die vorgestellten Lösungsansätze auf Kompatibilität geprüft werden, d.h. es muss geprüft werden, ob sich die Abbildungsmöglichkeiten einer Anforderung mit den Abbildungsmöglichkeiten einer anderen Anforderung vereinbaren lassen.

Zur **Abbildung von Anforderung 2** muss für jede Kontextvariable der Modus verwaltet werden. Für die Lösungsansätze zur Abbildung von Anforderung 1 bedeutet dies, dass neben dem Namen und dem aktuellen Wert der Kontextvariablen ein zusätzliches Attribut gepflegt werden muss. Dies lässt sich in allen drei vorgestellten Verwaltungsmöglichkeiten als Prozessdaten, in einer Datenbank sowie in externen Dateien realisieren. Bei der Verwaltung in einer relationalen Datenbank bspw. kann dies durch eine zusätzliche Spalte für den Modus erreicht werden.

Der Lösungsansatz zu Anforderung 2 fordert zusätzlich, dass Kontextvariablen mit Modus statisch zur Laufzeit gesperrt werden, damit Änderungen an ihnen keine Auswirkungen auf die Ausführung haben, da sie schon vor Prozessstart vorliegen müssen. Dementsprechend dürfen diese Kontextvariablen zur Laufzeit nicht verändert werden.

Prozessdaten können nicht explizit vor einem schreibenden Zugriff einer Aktivität geschützt werden. Hierfür müssten eigene Listener[1] für Datenobjekte implementiert werden, die bei einer Änderung aktiviert werden und bei Änderungen an statischen Kontextvariablen diese zurückweisen.

[1] Ein Listener wartet auf ein bestimmtes Ereignis und führt bei Eintreten desselben bestimmte Befehle aus.

In Datenbanken können Attribute auf verschiedenste Weise vor einem schreibenden Zugriff geschützt werden. So können bspw. spezielle Zugriffsrechte oder sog. Datenbank-Trigger [2] definiert werden. Durch die Zugriffsrechte kann der Zugriff auf die Datenquellen eingeschränkt und somit etwaige Änderungen verboten werden [Sch00]. Datenbank-Trigger können so implementiert werden, dass sie bei Änderung eines bestimmten Attributs aktiviert werden und die Änderung unterbinden [Dad96].

Um den Zugriffsschutz bei externen Dateien zu erreichen, können diese als schreibgeschützt definiert werden. Somit ist es nicht mehr möglich die Datei zu modifizieren. Allerdings können nur komplette Dateien und nicht einzelne Zeilen einer Datei schreibgeschützt werden. Demnach müssten zur Abbildung von Anforderung 2 die statischen Kontextvariablen in einer eigenen schreibgeschützten Datei verwaltet werden, während die dynamischen Kontextvariablen in einer weiteren, nicht schreibgeschützten, Datei verwaltet werden. In Tabelle 6.1 sind die Vor- und Nachteile zur Abbildung von Anforderung 2 in den drei vorgestellten Verwaltungsmöglichkeiten zusammengefasst.

Tabelle 6.1: Vor- und Nachteile bzgl. der Abbildung von Anforderung 2

Lösungsansatz	Vorteile	Nachteile
Prozessdaten	• Zusätzliches Attribut Modus ist umsetzbar	• Einzelne Datenobjekte können nicht explizit vor schreibendem Zugriff geschützt werden
Datenbank	• Zusätzliches Attribut Modus ist umsetzbar • Datenobjekte können schreibgeschützt werden	
Externe Datei	• Zusätzliches Attribut Modus ist umsetzbar • Datenobjekte können schreibgeschützt werden	• Zur Erreichung des Schreibschutzes müssen statische und dynamische Kontextvariablen separat voneinander verwaltet werden

Anhand der Vor- und Nachteile der einzelnen Verwaltungsmöglichkeiten bietet sich die Verwaltung der Kontextinformationen in einer Datenbank als Lösungsmöglichkeit zur Abbildung von Anforderung 2 an. Mit den Funktionalitäten einer Datenbank lässt sich sowohl ein

[2] Datenbank-Trigger sind im Datenbankmanagementsystem hinterlegte Prozeduren, die auf bestimmte Ereignisse reagieren [Dad96].

zusätzliches Attribut als auch der Zugriffsschutz für die statischen Kontextvariablen ohne Workaround, wie bspw. das Verwalten einer separaten Datei, realisieren.

Zur **Abbildung von Anforderung 3** ist entweder eine gemeinsame oder eine getrennte Verwaltung der Kontextinformationen möglich. Bei einer gemeinsamen Datenverwaltung muss bei etwaigen Änderungen an den globalen Kontextvariablen eine Aktualisierung der Werte bei allen betroffenen Prozessinstanzen erfolgen (vgl. Pattern 14 in Russell et al. [RHAE04]). Hierzu könnte die Prozessinstanz alle laufenden Prozessinstanzen des Prozesstyps über die Änderungen informieren (vgl. Beispiel 6.2). Allerdings kann es vorkommen, dass eine Vielzahl von Prozessinstanzen informiert wird, für die Änderungen nicht interessant sind, weil die Kontextvariable für deren Ausführung nicht relevant ist oder eine Auswertung schon an einem früheren Zeitpunkt erfolgt ist. Zudem müsste für jede Prozessinstanz eine entsprechende Verarbeitung der eintreffenden Information implementiert werden.

Beispiel 6.2 (Push-Prinzip) Abbildung 6.5 veranschaulicht das Push-Prinzip. Sobald eine Prozessinstanz eine globale Kontextvariable ändert, benachrichtigt sie alle betroffenen Prozessinstanzen über die Änderungen.

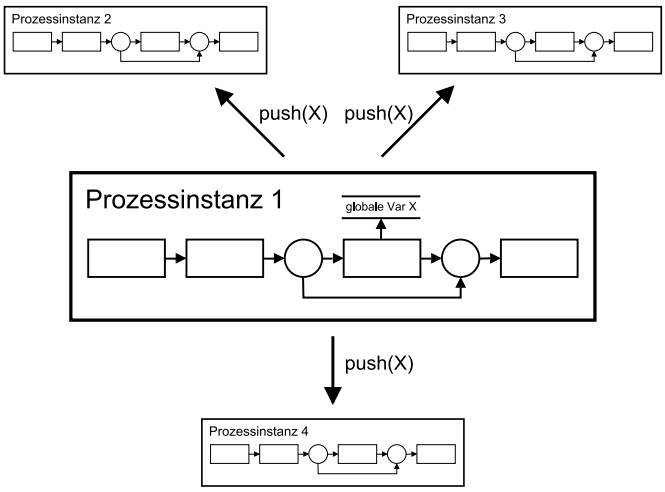

Abbildung 6.5: Push-Prinzip

Bei einer getrennten Verwaltung der lokalen und globalen Kontextinformationen pflegt jede Prozessinstanz ein Kontextmodell mit den lokalen Kontextvariablen und für alle Prozessinstanzen des Prozesstyps wird ein globales Kontextmodell zentral verwaltet. Hierbei kann

jede Prozessinstanz zu jeder Zeit auf das zentrale Kontextmodell zugreifen und es modifizieren. Benötigt eine Prozessinstanz den Wert einer globalen Kontextvariablen, so greift sie einfach an der entsprechenden Stelle während der Ausführung auf das Kontextmodell zu (vgl. Beispiel 6.3).

Beispiel 6.3 (Pull-Prinzip) Abbildung 6.6 skizziert das Pull-Prinzip. Sobald eine Prozessinstanz den Wert einer globalen Kontextvariablen ändern will, so hinterlegt sie den neuen Wert im zentral verwalteten globalen Kontextmodell. Alle weiteren Prozessinstanzen holen sich den neuen Wert, indem sie auf das zentrale Kontextmodell zugreifen.

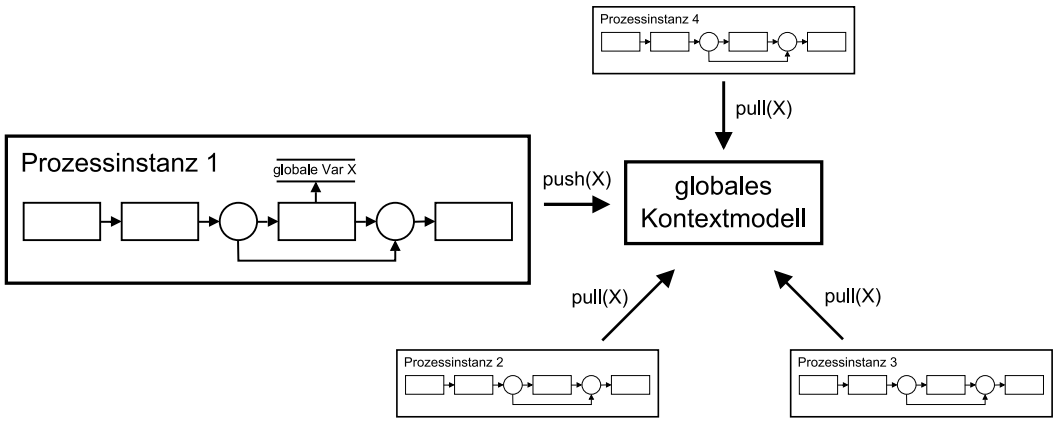

Abbildung 6.6: Pull-Prinzip

In Tabelle 6.2 sind die Vor- und Nachteile der gemeinsamen bzw. getrennten Verwaltung der Kontextinformationen zusammengefasst.

Anhand der Vor- und Nachteile der jeweiligen Ansätze stellt sich die getrennte Verwaltung der lokalen und globalen Kontextinformationen als die praktikabelste Lösung heraus, da bei Änderungen an globalen Kontextvariablen nur das zentral verwaltete globale Kontextmodell informiert werden muss und somit unnötige Benachrichtigungen der restlichen Prozessinstanzen entfallen. Nun muss evaluiert werden, wie die Abbildungsmöglichkeiten von Anforderung 1 mit diesem Ansatz umgehen können.

Die Verwaltung der Kontextinformationen als Prozessdaten ist nicht möglich, da die Prozessdaten immer nur für eine spezifische Prozessinstanz gelten. Die Prozessdaten sind somit nicht global und von außerhalb nur über die API-Schnittstelle manipulierbar.

Tabelle 6.2: Vor- und Nachteile bzgl. der gemeinsamen oder getrennten Verwaltung der Kontextinformationen

Lösungsansatz	Vorteile	Nachteile
Gemeinsame Verwaltung		• Informierung über Änderungen an globalen Kontextinformationen • Zusätzlicher Implementierungsaufwand zur Handhabung der Benachrichtigung
Getrennte Verwaltung	• Änderungen müssen nicht weitergegeben werden • Kein zusätzlicher Implementierungsaufwand zur Handhabung der Benachrichtigung	• Pflege eines zusätzlichen zentralen Kontextmodells für alle Prozessinstanzen

Die unterschiedliche Handhabung lokaler und globaler Kontextinformationen ist bei der Verwaltung in einer Datenbank ohne Weiteres möglich. Für jede Prozessinstanz gibt es eine lokale Datenquelle, bspw. eine Tabelle bei relationalen Datenbanken, und zudem gibt es eine zentrale Datenquelle zur Verwaltung der globalen Kontextvariablen. Im Zugriff auf die beiden Datenquellen gibt es keinen Unterschied.

Auch die Verwaltung der Kontextinformationen in externen Dateien lässt sich für lokale und globale Kontextvariablen realisieren. Jede Prozessinstanz hat eine Datei zum Verwalten der lokalen Daten und es existiert eine zentrale Datei zur Verwaltung der globalen Daten, auf die alle Prozessinstanzen zugreifen können. Die Zugriffsformen sind für beide Dateien gleich. In Tabelle 6.3 sind die Vor- und Nachteile der Verwaltungsmöglichkeiten zur Abbildung von Anforderung 1 aufgelistet.

Anhand der Vor- und Nachteile der jeweiligen Ansätze stellen sich die Verwaltung der Kontextinformationen in einer Datenbank sowie in externen Dateien als die praktikabelsten Lösungen heraus. Bei einer Datenbank muss zwar ein zusätzliches System gepflegt werden, jedoch können damit komfortabel Zugriffsrechte auf statische Kontextvariablen definiert werden. Des Weiteren kann mit einer Datenbank auch ein hohes Maß an Transaktionssicherheit bei konkurrierenden Zugriffen auf die Datenquellen sowie bei etwaigen Systemausfällen gewährleistet werden. Externe Dateien sind simpler einzurichten als Datenbanken und bieten ebenso die Möglichkeit, den Schreibschutz für statische Kontextvariablen

Tabelle 6.3: Vor- und Nachteile bzgl. der Verwaltung der Kontextinformationen

Lösungsansatz	Vorteile	Nachteile
Prozessdaten	• Keine eigene Erstellung von Zugriffsmethoden für die Daten	• Handhabung eines separaten globalen Kontextmodells ist nicht umsetzbar
Datenbank	• Separates globales Kontextmodell ist umsetzbar	• Pflege einer zusätzlichen Datenbank • Eigene Zugriffsmethoden auf die Datenbank sind zu implementieren
Externe Datei	• Globales Kontextmodell ist umsetzbar	• Pflege zusätzlicher Dateien • Eigene Zugriffsmethoden auf die Dateien sind zu implementieren

zu realisieren. Jedoch bringt die Verwaltung der Kontextinformationen in externen Dateien keinerlei Transaktionssicherheit mit sich. Bei parallelen Zugriffen auf eine Datei durch mehrere Prozessinstanzen kann es zu Fehlern kommen. Dementsprechend ist für die Verwaltung der Kontextinformationen der Lösungsansatz der Verwaltung in einer Datenbank zu priorisieren.

6.2 Abbildung der Optionsbeziehungen

Nachdem in Abschnitt 5.2 die spezifischen Anforderungen der Abbildung struktureller und semantischer Abhängigkeiten erläutert wurden, wird in diesem Abschnitt auf verschiedene Abbildungssmöglichkeiten eingegangen.

6.2.1 Lösungsansätze

Anforderung 4 fordert, dass die strukturellen und semantischen Abhängigkeiten zwischen den Optionen in einer maschinenlesbaren Form erfasst und verwaltet werden. Hierzu bietet sich die Verwaltung der Optionsbeziehungen als Prozessdaten, in einer Datenbank oder in externen Dateien an. Da diese Abbildungsmöglichkeiten bereits in Abschnitt 6.1.1 diskutiert wurden, entfällt an dieser Stelle eine Beschreibung derselben.

Gemäß **Anforderung 5** muss während der Ausführung eine Auswertung der Optionsbeziehungen erfolgen und somit die zulässigen Kombinationen für die Anwendung der Optionen ermittelt werden. Die Provop-Korrektheitsprüfung gewährleistet, dass es zur Laufzeit immer mindestens eine gültige Kombination von Optionen gibt. Widersprüchliche Optionsbeziehungen werden ausgeschlossen [HBR09b, HBR09c]. Um die vorhandenen Optionsbeziehungen auswerten zu können, müssen diese gelesen und verarbeitet werden. Wie dies geschieht, hängt von der Art der Verwaltung ab, d.h. ob als Prozessdaten, in einer Datenbank oder in externen Dateien. Ein wichtiges Kriterium dieser Anforderung ist der Zeitpunkt der Auswertung. Hierbei sind folgende Lösungsansätze möglich:

- **Auswertung bei Anwendung einer Option.** Die Optionsbeziehungen werden zur Laufzeit ausgewertet und zwar jedesmal, wenn über die Anwendung einer Option entschieden werden soll. Wird ein solcher Punkt während der Ausführung erreicht, muss auf Basis der Optionsbeziehungen sowie der bekannten Menge der bereits angewandten bzw. ausgelassenen Optionen entschieden werden, ob die aktuell betrachtete Option angewandt werden muss. In diesem Ansatz ist es nicht nötig, alle zulässigen Kombinationen der Optionen zu ermitteln, sondern nur über die Anwendbarkeit der aktuell betrachteten Option zu entscheiden.
- **Auswertung beim Start der Prozessinstanz.** Die Auswertung erfolgt bei Instanziierung des Prozessmodells. Dazu werden die beschriebenen Optionsbeziehungen gelesen und ausgewertet und die ermittelten, zulässigen Kombinationen von Optionen als Ergebnis gespeichert. Soll während der Ausführung über die Anwendung einer Option entschieden werden, so kann auf das Ergebnis der Auswertung zugegriffen werden. Die zulässigen Kombinationen können als Prozessdaten, in einer Datenbank oder in einer externen Datei gespeichert werden.

Neben dem Zeitpunkt der Auswertung ist auch die Art und Weise des Aufrufs der Auswertung interessant. Hierfür sind folgende Alternativen möglich:

- **Aufruf als externer Dienst.** Die Auswertung der Optionsbeziehungen wird aus dem Prozess heraus durch einen externen Dienst, bspw. einen Web-Service, erledigt. Hierzu muss der Dienst mit seinen Schnittstellen für den Zugriff definiert werden. Dem Dienst müssen als Eingabe die spezifizierten Abhängigkeiten sowie alle definierten Optionen mitgeteilt werden. Als Ergebnis liefert der Dienst entweder die zulässigen Kombinationen der Optionen (Auswertung beim Start der Prozessinstanz)

oder, ob eine entsprechende Option angewandt werden soll oder nicht (Auswertung bei Anwendung einer Option).

- **Aufruf als prozessinternes Skript.** Die Auswertung der Optionsbeziehungen wird von einem in den Prozess eingebundenen Skript durchgeführt. Das Skript wird in einer vom WfMS unterstützten Programmiersprache implementiert. Ein- und Rückgabeparameter des Skripts sind analog zu jenen des externen Dienstes.

Aus der Beschreibung der Lösungsansätze für Anforderung 5 wird ersichtlich, dass man für die Auswertungen der Optionsbeziehungen Informationen bzgl. der anzuwendenden Optionen benötigt. In den bisher diskutierten Abbildungsmöglichkeiten der Anforderungen wurde noch keine Verwaltung der Optionen vorgesehen. Die Verwaltungsmöglichkeiten sind analog zu jenen der Kontextinformationen sowie der Optionsbeziehungen. Für jede Option muss der Name der Option, eine textuelle Beschreibung sowie die zugehörige Kontextbedingung in Form von Prozessdaten, in einer Datenbank oder in externen Dateien verwaltet werden.

6.2.2 Diskussion der Abbildungsmöglichkeiten

Im Gegensatz zur Diskussion bzgl. der Kontextabhängigkeit müssen die Abbildungsmöglichkeiten für die Anforderungen durch Optionsbeziehungen nicht auf Kompatibilität geprüft werden, da die Art der Verwaltung sowie die Art der Auswertung der Optionsbeziehungen unabhängig voneinander sind.

Zur **Abbildung von Anforderung 4** müssen die strukturellen und semantischen Abhängigkeiten zwischen den Optionen verwaltet werden. Werden die Daten als Prozessdaten verwaltet, so müssen keine eigenen Zugriffsmethoden implementiert werden, da die gesamte Funktionalität schon im entsprechenden WfMS vorhanden ist. Die Prozessdaten werden dann zu Beginn der Ausführung für die Prozessinstanz angelegt. Die dabei spezifizierten Daten sind nur für die aktuelle Prozessinstanz gültig und können bei Bedarf nicht in anderen Prozessinstanzen wiederverwendet werden.

Bei der Verwaltung der Optionsbeziehungen in einer Datenbank können die Daten ggf. für die Ausführung einer anderen Prozessinstanz wiederverwendet werden. Allerdings stellt die Pflege einer zusätzlichen Datenbank einen Mehraufwand dar. Zudem müssen Zugriffsmethoden auf die Datenbank umgesetzt werden.

Werden die Optionsbeziehungen in externen Dateien verwaltet, so können die Daten ebenfalls wiederverwendet werden. Der Zugriff auf die Daten muss allerdings implementiert werden. In Tabelle 6.4 sind die Vor- und Nachteile der Lösungsansätze aufgelistet.

Tabelle 6.4: Vor- und Nachteile bzgl. der Verwaltung der Optionsbeziehungen

Lösungsansatz	Vorteile	Nachteile
Prozessdaten	• Keine eigene Erstellung von Zugriffsmethoden für die Daten	• Spezifizierte Daten gelten nur für die Prozessinstanz
Datenbank	• Wiederverwendbarkeit der Daten	• Pflege einer zusätzlichen Datenbank • Eigene Zugriffsmethoden auf die Datenbank sind zu implementieren
Externe Datei	• Wiederverwendbarkeit der Daten	• Pflege zusätzlicher Dateien • Eigene Zugriffsmethoden auf die Dateien sind zu implementieren

Anhand der Vor- und Nachteile der jeweiligen Ansätze stellt sich die Verwaltung der Optionsbeziehungen in allen drei Ansätzen als praktikabel heraus und lässt sich mit allen Ansätzen realisieren. Um die Komplexität der Abbildung jedoch möglichst gering zu halten, bietet sich am ehesten die Verwaltung als Prozessdaten an, da hierbei keine zusätzlichen Datenquellen wie etwa eine Datenbank gepflegt werden müssen und keine Zugriffsmethoden auf die extern liegenden Datenquellen implementiert werden müssen.

Zur **Abbildung von Anforderung 5** muss für die Auswertung der Optionsbeziehungen der geeignete Zeitpunkt sowie die Art und Weise des Aufrufs der Auswertung ermittelt werden. Erfolgt die Auswertung bei Anwendung einer Option während der Ausführung, so werden jedesmal die zulässigen Kombinationen der Optionen ermittelt. Bei einer hohen Anzahl von Optionen führt dies zu einem mehrmaligen Berechnen und somit zu einem zusätzlichen Rechenaufwand zur Laufzeit. Bei diesem Lösungsansatz müssen die Ergebnisse der Auswertung nicht gespeichert werden, da sie bei Anwendung einer Option neu ermittelt werden.

Findet die Auswertung der Optionsbeziehungen beim Start der Prozessinstanz statt, so erfolgt nur eine einmalige Berechnung, d.h. der Rechenaufwand wird minimiert. Werden die Ergebnisse der Auswertung zur Laufzeit benötigt, so muss ein Datenzugriff erfolgen. Al-

lerdings müssen hierfür die zulässigen Kombinationen entsprechend gespeichert werden. Hierfür bietet sich die Verwaltung als Prozessdaten an, da die Ergebnisse der Auswertung nur für diese eine Prozessinstanz relevant sind. In Tabelle 6.5 sind die Vor- und Nachteile der beiden Ansätze zusammengefasst.

Tabelle 6.5: Vor- und Nachteile bzgl. des Zeitpunktes der Auswertung der Optionsbeziehungen

Lösungsansatz	Vorteile	Nachteile
Auswertung zur Laufzeit	• Ergebnisse müssen nicht gespeichert werden, da sie jedesmal neu berechnet werden	• Rechenaufwand zur Laufzeit • Speicherung und Zugriff auf Optionsbeziehungen
Auswertung zu Prozessstart	• Einmaliger Rechenaufwand bei Prozessstart • Nur noch Datenzugriff zur Laufzeit	• Speicherung der Berechnungsergebnisse

Anhand der Vor- und Nachteile bietet sich die Auswertung der Optionsbeziehungen bei Start der Prozessinstanz an. Bei diesem Lösungsansatz ist der Rechenaufwand zur Berechnung der zulässigen Kombinationen konstant und unabhängig von der Anzahl der dynamischen Optionen. Dieser Ansatz ist insbesondere bei einer hohen Anzahl von dynamischen Optionen performanter.

Bei der Abbildung von Anforderung 5 muss zusätzlich die Art und Weise des Aufrufs der Auswertung berücksichtigt werden. Wird die Auswertung als externer Dienst umgesetzt, so ist die Umsetzung der Auswertung unabhängig von der im WfMS eingesetzten Programmiersprache und kann ggf. von mehreren unterschiedlichen WfMS verwendet werden. Allerdings ist die Spezifizierung der Schnittstellen für die Verwendung des Dienstes vergleichsweise aufwändig, bspw. durch Verwendung einer WSDL-Datei (vgl. [W3C07]). Da der Dienst zumeist auf einem externen Server läuft, besteht die Gefahr, dass der Server und somit auch der Dienst nicht immer verfügbar ist. Zudem kann es bei einer hohen Auslastung des Dienstes ggf. zu längeren Antwortzeiten kommen und die Ausführung der Prozessinstanzen verzögert sich.

Wird die Auswertung als prozessinternes Skript umgesetzt, benötigt es keine aufwändige Definition von Schnittstellenparametern wie bei einem externen Dienst. Allerdings ist man bei diesem Lösungsansatz an die im jeweiligen WfMS verwendete Programmiersprache gebunden. Zum Einsatz von Provop auf Basis unterschiedlicher WfMS, müsste das Skript

dann ggf. angepasst werden. Da das Skript prozessintern läuft, ist es für die Prozessinstanz sofort verfügbar. Zudem verkürzen sich bei diesem Ansatz die Antwortzeiten, da nicht auf den externen Dienst gewartet werden muss. In Tabelle 6.6 sind die Vor- und Nachteile der Ansätze zusammengefasst.

Tabelle 6.6: Vor- und Nachteile bzgl. der Art des Aufrufs für die Auswertung der Optionsbeziehungen

Lösungsansatz	Vorteile	Nachteile
Aufruf als Dienst	• Unabhängig von WfMS	• Aufwändige Spezifizierung der Schnittstelle für den Dienst • Ggf. geringe Verfügbarkeit • Längere Reaktionszeiten
Aufruf als Skript	• Einfache Spezifizierung des Aufrufs • Hohe Verfügbarkeit • Kurze Reaktionszeiten	• Abhängigkeit von Programmiersprache des WfMS

Anhand der Vor- und Nachteile der Lösungsmöglichkeiten wird ersichtlich, dass prinzipiell beide einsetzbar sind. Zur Verminderung des Overheads für die Implementierung sollte der Aufruf als prozessinternes Skript gewählt werden. Durch ein prozessinternes Skript können eine höhere Verfügbarkeit und kürzere Reaktionszeiten gewährleistet werden. Dadurch verkürzt sich die Laufzeit einer Prozessinstanz. Die Abhängigkeit von der verwendeten Programmiersprache des WfMS stellt kein entscheidendes Hindernis dar, da der Algorithmus zur Auswertung der Optionsbeziehungen keine komplexen proprietären Programmierkonstrukte erfordert und dementsprechend auch in verschiedenen Programmiersprachen umsetzbar ist.

Für die Verwaltung der Optionen in einer Datenquelle stellt sich die analoge Diskussion wie zu Anforderung 4 (vgl. Tabelle 6.4). Wie in der dortigen Diskussion sind alle drei Lösungsmöglichkeiten praktikabel. Da die definierten Optionen ebenfalls nur für die laufende Prozessinstanz relevant sind und die Komplexität minimiert werden soll, bietet sich die Verwaltung als Prozessdaten an.

6.3 Abbildung der Provop-spezifischen Kontrollfluss-Konstrukte

Nachdem in Abschnitt 5.3 die spezifischen Anforderungen an die Abbildung der Aufsplittung und Synchronisation des Kontrollflusses erläutert wurden, werden in diesem Abschnitt entsprechende Lösungsmöglichkeiten vorgestellt.

6.3.1 Lösungsansätze

Anforderung 6 fordert, dass bei der Aufsplittung des Kontrollflusses über die Anwendbarkeit einer Option entschieden wird. Dabei ist relevant, in welcher Form die Prüfung erfolgt. Dabei sind, wie bei Anforderung 5, der Aufruf eines externen Dienstes oder eines prozessinternen Skripts denkbar. Da diese Abbildungsmöglichkeiten schon im vorhergehenden Abschnitt diskutiert wurden, wird an dieser Stelle auf eine Diskussion verzichtet.

Anforderung 7 sieht die Gewährleistung der Atomarität von Optionen vor. Dies kann durch Prüfung der Anwendbarkeit einer Option und anschließende Speicherung des Resultates erreicht werden. Soll bspw. bei Erreichen einer zweiten Änderungsoperation einer Option während der Ausführung über ihre Anwendung entschieden werden, so muss nur noch das gespeicherte Resultat gelesen werden und abhängig vom Resultat die Änderungsoperation angewandt oder ausgelassen werden. Auf diese Weise kann sichergestellt werden, dass immer alle oder gar keine Änderungsoperationen einer Option angewandt werden.

Das Ergebnis der Auswertung über die Anwendbarkeit einer Option muss für jede Option verwaltet werden. Hierzu ist denkbar, für jede Option ein zusätzliches Attribut *Status* zu pflegen, wobei folgende Zustände sinnvoll sind:

- **undefined.** Über die Anwendung einer dynamischen Option wurde noch nicht entschieden und somit auch noch keine ihrer Änderungsoperation angewandt oder ausgelassen.
- **apply.** Die dynamische Option soll angewandt werden.
- **skip.** Die dynamische Option soll ausgelassen werden.

Abbildung 6.7 zeigt das Zustandsdiagramm einer Option. Der initiale Zustand einer Option ist undefined und wird nach der Auswertung auf skip gesetzt, wenn die Option nicht ange-

wandt werden soll bzw. darf. Andernfalls wird ihr Zustand auf apply gesetzt. Ein Rücksetzen des Zustands zu undefined ist in keinem Fall möglich.

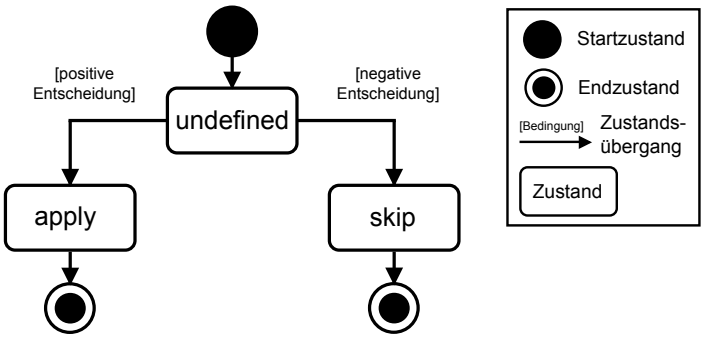

Abbildung 6.7: Zustandsdiagramm der Optionen

Gemäß **Anforderung 8** muss am OPJoin-Knoten eine korrekte Synchronisation erfolgen. Unerwünschte Ausführungsreihenfolgen wie in Beispiel 5.2 werden dadurch ausgeschlossen. Zur Gewährleistung der korrekten Synchronisation sind folgende Möglichkeiten denkbar:

- **Verwendung spezieller Synchronisationskanten.** Reichert [Rei00] führt in ADEPT sog. *Synchronisationskanten (kurz Sync-Kanten)* ein. Mit diesem Konstrukt lassen sich bei ADEPT Reihenfolgebeziehungen zwischen Aktivitäten paralleler Zweige modellieren. Existiert eine Sync-Kante von Aktivität A nach Aktivität B, so kann B frühestens dann ausgeführt werden, wenn A entweder zuvor erfolgreich beendet wurde oder wenn feststeht, dass A nicht mehr zur Ausführung kommen kann.

  Die in Anforderung 8 geforderte Semantik zur Gewährleistung einer korrekten Synchronisation entspricht der Semantik der Sync-Kante. Damit wird ein expliziter OPJoin-Strukturknoten zur Synchronisation in ADEPT hinfällig. Abbildung 6.8 zeigt ein Prozessfragment mit einer Sync-Kante.

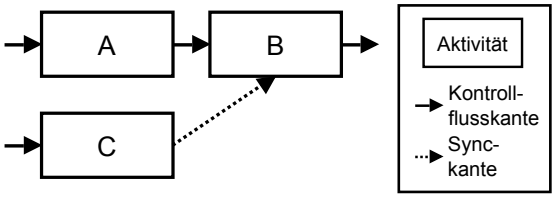

Abbildung 6.8: Korrekte Synchronisation mittels Sync-Kante

- **OPJoin mit Eingangsbedingung.** Zur Synchronisation wird für den OPJoin-Knoten eine spezielle Eingangsbedingung definiert, welche die Belegungen der eingehenden Kanten ermittelt und aufgrund dessen den ausgehenden Pfad aktivieren oder abwählen kann. Die Eingangsbedingung evaluiert zu `true`, falls alle eingehenden Kanten ein Signal tragen, d.h. nicht `undefined` sind und alle eingehenden regulären Kontrollflusskanten `true` signalisieren. Ist keine reguläre Kontrollflusskante gegeben, so muss mindestens eine Operationskante `true` signalisieren. Durch diese Eingangsbedingung wird die reguläre Kante stärker gewichtet und falls diese abgewählt wurde, kann auch keine aktivierte Operationskante eine Aktivierung des abgewählten Pfades bewirken.

  Die Eingangsbedingung kann auch durch die Verwendung von sog. *ECA-Regeln*[3] abgebildet werden. Die regelbasierte Beschreibungstechnik zur Beschreibung von Prozessen ist sehr ausdrucksstark, allerdings sind die Spezifizierungen sehr komplex und schlecht wartbar.

  Abbildung 6.9 zeigt relevante Fallbeispiele der Synchronisation am OPJoin-Knoten und die zugehörigen Ergebnissen der Evaluation der Eingangsbedingung.

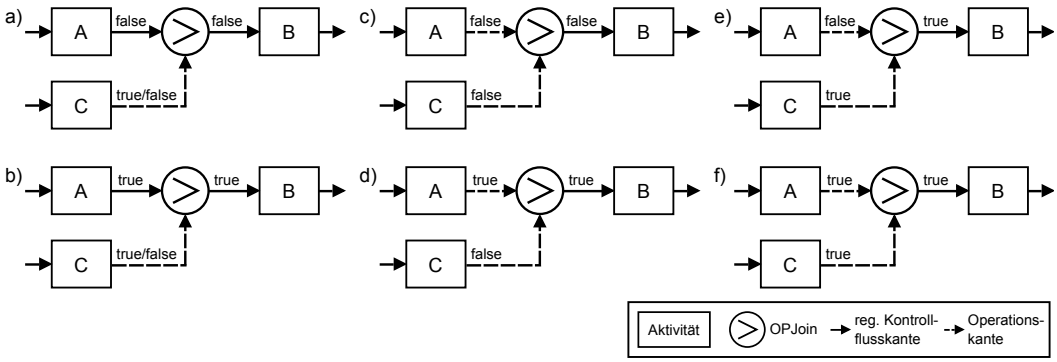

Abbildung 6.9: Fallbeispiele der korrekten Synchronisation am OPJoin-Knoten

6.3.2 Diskussion der Abbildungsmöglichkeiten

Zur **Abbildung von Anforderung 6** ist relevant, wie die Anwendbarkeit einer Option geprüft wird. Analog zu Anforderung 5 ist der Aufruf eines externen Dienstes oder eines prozessinternen Skriptes denkbar. Dementsprechend wird auf die obige Diskussion der Vor- und

[3] ECA steht für Event-Condition-Action und bedeutet, dass bei Auftreten des Ereignisses E und unter Einhaltung der Bedingung C die Aktion A ausgeführt wird [KEP00].

Nachteile verwiesen (vgl. Tabelle 6.6). Als Abbildungsmöglichkeit wird wie im vorigen Abschnitt das prozessinterne Skript gewählt. Anhand des Ergebnisses der Prüfung wird dann die entsprechende Aufsplittung während der Ausführung durchgeführt.

Zur **Abbildung von Anforderung 7** muss für jede Option ein zusätzliches Attribut gepflegt werden, welches den Status der Option speichert. Dementsprechend muss die Struktur der Verwaltungsmöglichkeit, welche in Abschnitt 6.2.2 als geeignete Abbildungsmöglichkeit zur Verwaltung der Optionen gewählt wurde, entsprechend erweitert werden.

Zur **Abbildung von Anforderung 8** ist entweder die Verwendung spezieller Sync-Kanten oder die Spezifizierung einer Eingangsbedingung im OPJoin-Knoten zur Auswertung der Kantenbelegungen denkbar. Erfolgt die Synchronisation der Pfade durch Sync-Kanten, so entfällt der zusätzliche OPJoin-Knoten. Die Semantik der Sync-Kante impliziert die spezielle Eingangsbedingung des OPJoin-Knotens von Provop. Allerdings muss dieser neue Kantentyp mit seiner Semantik in den WfMS definiert und umgesetzt werden bzw. realisierbar sein.

Erfolgt die Synchronisation durch die Verwendung der Eingangsbedingung am OPJoin-Knoten, so muss dieser Strukturknoten in das Prozessmodell zur Synchronisation der Pfade integriert werden. Der Zugriff auf die Belegungen der Kanten zur Auswertung der Eingangsbedingung ist in jedem WfMS möglich, da diese Informationen Teil der Ausführungshistorie eines jeden Prozesses sind. Das Ergebnis der Auswertung muss dann als Bedingung der ausgehenden Kante überprüft werden. Dies ist jedoch nicht in WfMS möglich, die nur simple XOR-,OR- oder ANDJoins anbieten. In Tabelle 6.7 sind die Vor- und Nachteile der Ansätze zusammengefasst.

Tabelle 6.7: Vor- und Nachteile bzgl. der Art des Aufrufs für die Synchronisation mit spezieller Eingangsbedingung

Lösungsansatz	Vorteile	Nachteile
Verwendung der Sync-Kante	• Kein zusätzlicher Strukturknoten für die Synchronisation erforderlich • Implizite Auswertung der Kantenbelegungen	• ggf. Einführung eines neuen Kantentyps
OPJoin mit Eingangsbedingung	• Ermittlung der Belegungen der Kanten ist möglich	• Zusätzlicher Strukturknoten • Explizite Auswertung der Kantenbelegungen

Anhand der Vor- und Nachteile der beiden Ansätze stellt sich die Verwendung eines OPJoin-Knotens mit Eingangsbedingung als passender heraus. Zwar erfordert dieser Ansatz einen zusätzlichen Knoten zur Synchronisation, die Umsetzung der Auswertung lässt sich aber in allen WfMS realisieren, bei denen für ausgehende Kanten Bedingungen definiert werden können. Die Sync-Kante würde den Strukturknoten im Prozessmodell ersparen, jedoch gibt es diesen Kantentyp nur in ADEPT und müsste bei einer Umsetzung des Provop-Lösungsansatzes in anderen WfMS noch definiert werden.

6.4 Diskussion der Umsetzung in den WfMS

In diesem Abschnitt wird diskutiert, wie die in Abschnitt 4.2 vorgestellten WfMS ADEPT, Tibco iProcess Suite, WebSphere Integration Developer und YAWL-System die in diesem Kapitel erläuterten Abbildungsmöglichkeiten der Anforderungen unterstützen.

Gemäß der Lösungsmöglichkeit zu **Anforderung 1** sollten die Kontextinformationen in einer Datenbank oder in externen Dateien verwaltet werden. Die Verwaltung in einer Datenbank ist in allen WfMS möglich. Für den Zugriff auf die Schnittstellen der verschiedenen Datenbanken gibt es in allen WfMS entsprechenden Mechanismen, bspw. für das Laden von Treibern. Die Abfragen für den Datenzugriff, wie etwa eine SQL-Query, müssen in den Systemen hinterlegt werden. Auch die als Alternative vorgesehene Verwaltung der Kontextinformationen in externen Dateien ist in den betrachteten WfMS umsetzbar.

Für die **Anforderungen 2 und 7** wird als Lösungsmöglichkeit jeweils die Verwaltung eines zusätzlichen Attributs in der verwendeten Datenstruktur gefordert. Sie stellen somit nur eine Ergänzung der gewählten Verwaltungsmöglichkeit dar. Demnach sind diese Lösungsmöglichkeiten unabhängig von den WfMS und werden in diesem Abschnitt nicht betrachtet.

Für die **Anforderung 3** wird als Lösungsmöglichkeit die getrennte Verwaltung der lokalen und globalen Kontextinformationen gefordert. Dies hat keinen direkten Einfluss auf die Umsetzung in den verschiedenen WfMS. Durch eine getrennte Verwaltung erhöht sich nur die Anzahl der Datenquellen, auf die das WfMS zugreifen muss. Dementsprechend ist auch diese Lösungsmöglichkeit unabhängig von den WfMS und wird in dieser Diskussion ebenfalls nicht berücksichtigt.

Gemäß der Abbildungsmöglichkeit zu **Anforderung 4** sollten die strukturellen und semantischen Abhängigkeiten zwischen Optionen als Prozessdaten verwaltet werden. Des Wei-

teren wird in Abschnitt 6.2.2 darauf hingewiesen, dass auch die definierten Optionen als Prozessdaten verwaltet werden sollten. In allen betrachteten WfMS gibt es die Möglichkeit zur Verwaltung dieser Daten als prozessinterne Daten, d.h. es können zur Verwaltung entsprechende Datenstrukturen definiert werden, im WebSphere Integration Developer bspw. durch die Definition sog. *Business Objects*. Diese Datenobjekte können aus ein oder mehreren Attributen bestehen und Werte von gewissen Datentypen annehmen. Die Datenobjekte können auch geschachtelt werden. Die Speicherung der Datenobjekte erfolgt, wie auch in YAWL, im XML-Format.

Als Lösungsansatz von **Anforderung 5** wird der Aufruf eines prozessinternen Skripts nach der Instanziierung des Prozessmodells vorgeschlagen. Dieses Skript kann als zusätzlich in das Prozessmodell eingebundene Aktivität dargestellt werden. Als Eingabeparameter erhält die Aktivität die für die Auswertung erforderlichen Datenquellen und generiert als Ausgabe die zulässigen Kombinationen von Optionen. Die Abbildung des Skripts auf eine zusätzlich Aktivität, die sofort nach Prozessstart ausgeführt wird, ist in allen betrachteten WfMS möglich, z.B. können im WebSphere Integration Developer hierfür sog. Java Snippet Activities in das Prozessmodell eingebunden werden, welche die Auswertung in Form eines Java-Programms erledigen.

Gemäß der Abbildungsmöglichkeit zu **Anforderung 6** erfolgt die Prüfung der Anwendbarkeit einer dynamischen Option durch ein prozessinternes Skript. Die Prüfung der Anwendbarkeit erfolgt generell wenn eine dynamische Option angewandt werden soll. Dementsprechend wird an dieser Stelle im Prozessmodell eine entsprechende Aktivität benötigt, die über die Anwendbarkeit einer dynamischen Option entscheidet. Das Einfügen einer solchen Aktivität mit entsprechendem Aufruf eines Skriptes ist in allen betrachteten WfMS möglich.

In der Diskussion zu den Abbildungsmöglichkeiten zu Anforderung 5 und 6 in den Abschnitten 6.2.2 und 6.3.2 wird der Aufruf eines externen Dienstes als Alternative zum Aufruf eines prozessinternen Skripts vorgeschlagen, der dann die entsprechenden Auswertungen bzw. Prüfungen erledigt. Auch diese Lösungsmöglichkeit lässt sich in den betrachteten WfMS realisieren. An jener Stelle, wo die Aktivitäten für das prozessinterne Skript in das Prozessmodell integriert werden, erfolgt dann der Aufruf des Dienstes. In YAWL werden externe Dienste bspw. über den *YAWL-Webservice-Broker* integriert [AADH04].

Zur Abbildung von **Anforderung 8** wird als Lösungsmöglichkeit die Verwendung eines OPJoin-Knotens mit spezieller Eingangsbedingung vorgeschlagen. Dieser Lösungsansatz

wird von keinem der WfMS direkt unterstützt. Einzig ADEPT bietet mit den Sync-Kanten eine vergleichbare Lösungsmöglichkeit (vgl. Abschnitt 6.3.2). Jedoch kann der OPJoin-Knoten als explizite Aktivität in das Prozessmodell integriert werden und die spezielle Eingangsbedingung intern überprüfen und somit den weiteren Kontrollfluss steuern. Dieses Vorgehen kann in jedem der betrachteten WfMS mit entsprechenden Workarounds umgesetzt werden.

In Tabelle 6.8 sind die Ergebnisse der Diskussion bzgl. der Umsetzung der Abbildungsmöglichkeiten zusammengefasst. Ein "+" bedeutet, dass das WfMS die Abbildung der Anforderung umsetzen kann. Ein "0" bedeutet, dass die Abbildung dieser Anforderung durch einen Workaround realisiert werden kann, während ein "-" bedeutet, dass die Anforderung nicht in dem WfMS abgebildet werden kann.

Tabelle 6.8: Diskussion über Abbildung der Anforderungen in den WfMS

Anforderung\WfMS	ADEPT	Tibco	WID	YAWL
Verwaltung der Kontextinformationen in einer Datenbank	+	+	+	+
Verwaltung der Optionsbeziehungen und Optionen als Prozessdaten	+	+	+	+
Aufruf eines prozessinternen Skripts zur Auswertung der Optionsbeziehungen nach Instanziierung des Prozessmodells	+	+	+	+
Aufruf eines prozessinternen Skripts zur Prüfung der Anwendbarkeit von Optionen	+	+	+	+
Synchronisation mit Eingangsbedingung	0	0	0	0

6.5 Zusammenfassung

In diesem Kapitel werden geeignete Lösungsansätze für die Abbildung der Anforderungen der Provop-Konzepte diskutiert. Zur Abbildung jeder Anforderung werden verschiedene Lösungsansätze vorgestellt, deren Vor- und Nachteile erörtert und eine Präferenz für die Abbildungsmöglichkeit angegeben. Dabei wird erläutert, dass Kontextvariablen anhand lokaler und globaler Geltungsbereiche unterschieden werden und deren Verwaltung in einer Datenbank erfolgen sollte. Optionsbeziehungen sowie Optionen hingegen sollten während der Ausführung als Prozessdaten verwaltet werden.

Die Auswertung der Optionsbeziehungen während der Ausführung erfolgt durch den Aufruf eines prozessinternen Skripts bei Start der Prozessinstanz. Ebenso erfolgt die Prüfung der

Anwendbarkeit von dynamischen Optionen durch den Aufruf eines prozessinternen Skripts bei Anwendung einer dynamischen Option. Die Atomarität von Optionen wird durch die Einführung eines zusätzlichen Status für die Optionen gewährleistet.

Die korrekte Synchronisation der Ausführungspfade nach Anwendung einer dynamischen Option wird durch die Integration eines OPJoin-Knotens mit spezieller Eingangsbedingung in das Prozessmodell zugesichert. Dadurch werden fehlerhafte Ausführungsreihenfolgen verhindert.

In Tabelle 6.9 sind nochmals alle Anforderungen der Provop-Konzepte mit dem gewählten Lösungsansatz aufgelistet.

Tabelle 6.9: Gewählte Lösungsansätze zur Abbildung der Anforderungen der Provop-Konzepte

Anforderung 1: Verwaltung der Kontextinformationen. Lösung: Die Kontextinformationen werden in einer Datenbank verwaltet.
Anforderung 2: Aktualisierung von Kontextvariablen. Lösung: Für jede Kontextvariable wird ein zusätzliches Attribut *Modus* verwaltet, welches die Werte *statisch* oder *dynamisch* annehmen kann.
Anforderung 3: Geltungsbereich der Kontextinformationen. Lösung: Für die Unterscheidung zwischen lokalen und globalen Kontextinformationen werden diese in unterschiedlichen Datenquellen verwaltet.
Anforderung 4: Verwaltung der Optionsbeziehungen. Lösung: Die strukturellen und semantischen Abhängigkeiten zwischen den Optionen werden als Prozessdaten verwaltet.
Anforderung 5: Auswertung der Optionsbeziehungen. Lösung: Die Abhängigkeiten zwischen den Optionen werden von einem prozessinternen Skript nach dem Start der Prozessinstanz ausgewertet und dabei die zulässigen Kombinationen von Optionen berechnet.
Anforderung 6: Prüfung der Anwendbarkeit einer dynamischen Option. Lösung: Vor der Anwendung dynamischer Änderungsoperationen erfolgt die Prüfung durch den Aufruf eines prozessinternen Skripts.
Anforderung 7: Gewährleistung der Atomarität von Optionen. Lösung: Zur Gewährleistung der Atomarität von Optionen wird für jede Option ein zusätzliches Attribut *Status* gepflegt, das besagt, ob die Option bereits angewandt wurde oder nicht. Das Attribut kann die Werte *undefined*, *apply* oder *skip* annehmen.
Anforderung 8: Gewährleistung der korrekten Synchronisation. Lösung: Zur Gewährleistung der korrekten Synchronisation erfolgt diese durch einen OPJoin-Knoten mit spezieller Eingangsbedingung.

Zusätzlich wird für jeden der gewählten Lösungsansätze diskutiert, ob und in welcher Form dieser in den betrachteten WfMS umsetzbar ist. Wurden bei der Diskussion der verschie-

denen Lösungsansätze Alternativen vorgeschlagen, so werden auch diese in die Analyse miteinbezogen. Die Analyse der WfMS soll bei der Auswahl eines Systems für die praktische Umsetzung des Provop-Ansatzes im folgenden Kapitel helfen.

7 Umsetzung der Provop-Konzepte in einem WfMS

In Kapitel 6 wurden Abbildungsmöglichkeiten für die Anforderungen der Provop-Konzepte aus Kapitel 5 vorgestellt. In diesem Kapitel wird diskutiert, welches WfMS für die prototypische Umsetzung des Provop-Lösungsansatzes zur Ausführung von Prozessvarianten gewählt wird und in welcher Form die Umsetzung erfolgt.

Abschnitt 7.1 erläutert die Festlegung auf ein bestimmtes WfMS zur Umsetzung des Provop-Lösungsansatzes und stellt wichtige Funktionalitäten des gewählten WfMS vor. Abschnitt 7.2 diskutiert, wie die in Kapitel 6 erörterten Lösungsansätze zur Abbildung der Provop-Konzepte im gewählten WfMS umgesetzt werden. Abschnitt 7.3 gibt einen Überblick über die Gesamtarchitektur des Prototypen. Abschnitt 7.4 beschreibt anhand eines Fallbeispiels die Verwendung des Prototypen.

7.1 Wahl des WfMS zur prototypischen Umsetzung des Provop-Lösungsansatzes

In Abschnitt 6.4 wird beschrieben, wie die in Abschnitt 4.2 betrachteten WfMS die Abbildungsmöglichkeiten der Anforderungen der Provop-Konzepte unterstützen. Dabei wird ersichtlich, dass eine Umsetzung des Provop-Lösungsansatzes prinzipiell in allen Systemen möglich ist. Jedoch muss für die konkrete Implementierung der Konzepte ein System ausgewählt werden.

Der WebSphere Integration Developer von IBM ist bei der Daimler AG im Bereich der Geschäftsprozessmodellierung ein strategisches Werkzeug. Dieses Werkzeug ist ein mächtiges System und bietet mit dem WebSphere Process Server eine integrierte Testumgebung

zur Ausführung der Prozessvarianten an. Zudem wird im WebSphere Integration Developer die Prozessbeschreibungssprache BPEL eingesetzt, welche einen wichtigen Standard im Bereich der Prozessmodellierung darstellt. Dementsprechend erfolgt die prototypische Umsetzung des Provop-Lösungsansatzes im Rahmen dieser Arbeit im WebSphere Integration Developer. Abbildung 7.1 zeigt die graphische Benutzeroberfläche des Werkzeugs.

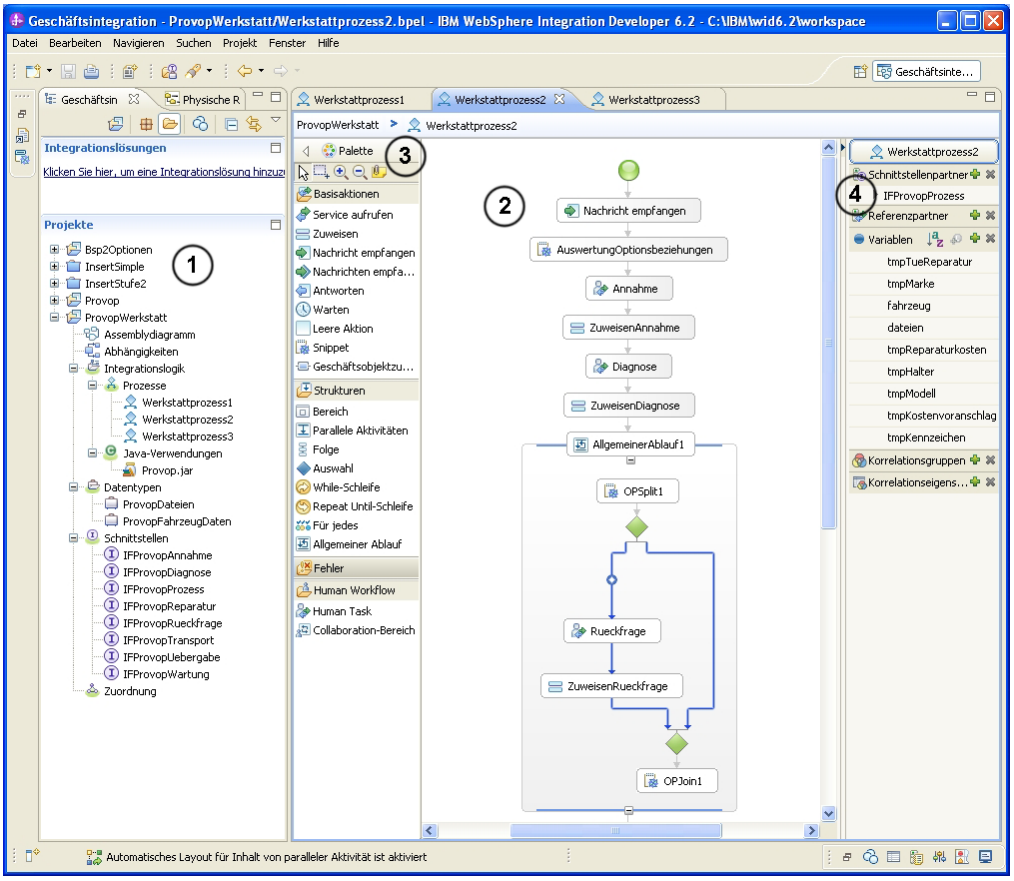

Abbildung 7.1: Graphische Benutzeroberfläche des WebSphere Integration Developer

In (1) werden alle Projekte des aktuellen Arbeitsbereiches mit all ihren zugehörigen Elementen, wie bspw. den definierten Prozessmodellen, angezeigt. (2) stellt die Zeichenfläche dar, auf der die Prozessmodelle modelliert werden können.[1] Die Repräsentation des Prozessmodells erfolgt graphisch, kann aber wahlweise auch auf textuell umgestellt werden. Die Palette (3) enthält Elemente, die zur Modellierung der Prozesse verwendet

[1] In der Abbildung ist dort ein Beispiel des Werkstattprozesses abgebildet.

werden können. Neben Standard-BPEL-Konstrukten enthält die Palette auch die BPEL-Erweiterungen von IBM, wie bspw. Java Snippet Activities. Alle Elemente der Palette können per Drag&Drop auf die Zeichenfläche gezogen werden. (4) dient zur Anzeige und Bearbeitung von prozess-spezifischen Informationen für das ausgewählte Prozessmodell, wie bspw. der Prozessdaten.

Die Modellierung von Geschäftsprozessen kann im WebSphere Integration Developer mittels BPEL-Prozessen (vgl. Abschnitt 2.2.3) oder Statusmaschinen (vgl. [BC06]) erfolgen. Wie schon bei der Beschreibung der graphischen Benutzeroberfläche erwähnt, können BPEL-Prozesse in einem graphischen Editor modelliert werden. Für die Modellierung von Prozessmodellen gibt es neben den Sprach-Konstrukten von BPEL einige proprietären Erweiterungen des BPEL-Standards. Zur prototypischen Umsetzung der Provop-Konzepte sind dabei folgende Erweiterungen interessant:

- **Java Snippet Activities.** Sie erlauben die Ausführung von beliebigem Java-Code innerhalb des Prozessablaufs. Durch sie kann bspw. auf Prozessdaten, Ausführungszustände von Aktivitäten oder auch auf externe Ressourcen, wie etwa Dateien, zugegriffen werden.

- **Allgemeiner Ablauf.** Dieses Konstrukt ermöglicht eine graph-orientierte Modellierung in einem BPEL-Prozess. In einen “Allgemeinen Ablauf“ können beliebige BPEL-Konstrukte über *Links* miteinander verbunden werden. Damit wird der Kontrollfluss zwischen den Aktivitäten bestimmt. Wird im Allgemeinen Ablauf eine Verzweigung modelliert, so wird automatisch eine Verknüpfung der Links erzeugt. Dabei kann die Semantik der Verzweigung festgelegt werden, wobei AND-, OR- sowie XOR-Verzweigung möglich sind. Abbildung 7.2 zeigt einen einfachen Allgemeinen Ablauf mit einer OR-Verzweigung nach dem Java Snippet.

Mittels Statusmaschinen werden Geschäftsprozesse beschrieben, die klar definierte Zustände durchlaufen und deren Verhalten vom aktuellen Zustand abhängt. Für die prototypische Umsetzung der Provop-Konzepte beschränken wir uns auf die Modellierung mittels BPEL. Dementsprechend werden Statusmaschinen nicht näher betrachtet.

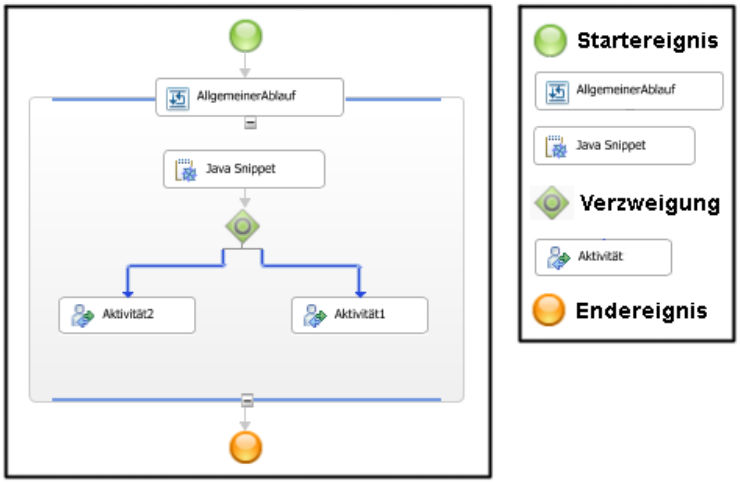

Abbildung 7.2: Beispiel für einen Allgemeinen Ablauf

7.2 Prototypische Umsetzung

Nachdem im vorhergehenden Abschnitt der WebSphere Integration Developer vorgestellt wurde, geht dieser Abschnitt darauf ein, wie die gewählten Abbildungsmöglichkeiten für die Umsetzung des Provop-Lösungsansatzes zur Ausführung von Prozessvarianten in diesem Werkzeug implementiert werden. Dabei werden die Verwaltung der Kontextinformationen, der Optionsbeziehungen, der Optionen, die Auswertung der Optionsbeziehungen, die Prüfung der Anwendbarkeit der Optionen sowie die korrekte Synchronisation der Ausführungspfade von Option und Basisprozess betrachtet. Für jeden Aspekt der Datenverwaltung wird erläutert, welcher Lösungsansatz zur prototypischen Umsetzung verwendet wird, wie eine geeignete Datenstruktur zur Repräsentation der Daten aussehen kann und wie der Zugriff auf die gespeicherten Daten erfolgt. Für die Auswertung der Optionsbeziehungen sowie für die Prüfung der Anwendbarkeit der dynamischen Optionen wird der generelle Ablauf des jeweiligen Algorithmus und die Umsetzung im WebSphere Integration Developer erläutert.

7.2.1 Verwaltung der Kontextinformationen

Zur Verwaltung der Kontextinformationen wird in Kapitel 6 die Verwaltung in einer Datenbank vorgeschlagen. Das Einrichten und Betreiben einer relationalen Datenbank samt Erstellen der Tabellen und Zugriffsregeln stellt bzgl. der prototypischen Umsetzung der

Provop-Konzepte jedoch einen unnötig hohen Aufwand dar. Dementsprechend wird für die Verwaltung der Kontextinformationen der Ansatz externer Dateien verfolgt.

Für die Verwaltung in externen Dateien sind verschiedene Datenformate denkbar, wie bspw. CSV oder XML. Mittels XML können komplexe Datenstrukturen abgebildet werden. Durch XML Schema kann eine Struktur für die Speicherung vorgegeben und somit ein entsprechendes Datenformat definiert werden. Da die in einem BPEL-Prozess definierten Prozessdaten ebenfalls mittels XML Schema spezifiziert werden, wird für die Verwaltung der Kontextinformationen XML als Datenformat gewählt.

Um eine geeignete Datenstruktur für die Verwaltung der Kontextinformationen in einer XML-Datei definieren zu können, muss zunächst ermittelt werden, wie in Provop der aktuell gegebene Kontext spezifiziert wird (vgl. Abschnitt 3.4.1). Ein Kontextmodell zur Erfassung der Kontextinformationen besteht aus mehreren Kontextvariablen. Dabei besteht eine Kontextvariable aus den folgenden Attributen:

- **Name.** Spezifiziert den Namen der Kontextvariable. Dieser ist in Provop eindeutig und dient somit zur Identifikation der Kontextvariablen.
- **Wert.** Stellt den Wert der Kontextvariable aus dem gegebenen Wertebereich dar.
- **Modus.** Gibt an, ob es sich um eine statische oder dynamische Kontextvariable handelt (vgl. Anforderung 2 in Kapitel 5).
- **Beschreibung.** Dient für eine etwaige Beschreibung der Kontextvariable.

Listing 7.1 zeigt die komplette Spezifikation eines Kontextmodells in XML Schema. Es besteht aus beliebig vielen Kontextvariablen, welche wiederum aus den obigen Attributen bestehen. Anhand der Schema-Spezifikation für Kontextvariablen kann eine valide XML-Datei erstellt werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="kontextmodell">
  <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="kontextvariable" type="kontextvariableTyp" maxOccurs="
         unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
```

```
</xsd:element>

<xsd:complexType name="kontextvariableTyp">
  <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="wert" type="xsd:anyType"/>
  <xsd:element name="modus">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="statisch"/>
        <xsd:enumeration value="dynamisch"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="beschreibung" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

Listing 7.1: XSD Schema des Kontextmodells

Die Verarbeitung der XML-Dateien erfolgt in Java Snippets durch Verwendung der JDOM-API.[2] Listing 7.2 zeigt exemplarisch einen Programmausschnitt aus einem Java Snippet für das Verarbeiten einer XML-Datei mittels JDOM. Das Kontextmodell wird eingelesen und anschließend die Attribute jeder Kontextvariable auf der Standard-Ausgabe ausgegeben. Abschließend wird das Dokument wieder im XML-Format in eine Datei geschrieben.

```
//Einlesen des Kontextmodells
Document doc = new SaxBuilder().build("kontextmodell.xml");
Element root = doc.getRootElement();
//Manipulieren des Inhalts; Hier werden alle Attribute gelesen und ausgegeben
List<?> listKtxtVariable = root.getChildren("kontextvariable");
for(int i=0; i<listKtxtVariable.size(); i++) {
  Element ele = (Element)(listKtxtVariable.get(i));
  if(ele == null) { continue; }
  System.out.println("Name: "+ele.getChild("name").getValue());
  System.out.println("Wert: "+ele.getChild("wert").getValue());
  System.out.println("Modus: "+ele.getChild("modus").getValue());
  System.out.println("Beschreibung: "+ele.getChild("beschreibung").getValue());
```

[2] JDOM ist eine API zum Einlesen, Manipulieren und Schreiben von XML-formatierten Dateien in Java [Ull09].

```
}
//Schreiben des XML-Baumes in eine Datei
XMLOutputter fmt = new XMLOutputter();
fmt.setFormat(Format.getPrettyFormat());
fmt.output( doc, new FileOutputStream(new File("kontextmodell.xml")));
```

Listing 7.2: Verarbeitung des Kontextmodells mittels JDOM

Wie in Abschnitt 6.1.1 diskutiert, sollen die lokalen und globalen Kontextinformationen getrennt voneinander in unterschiedlichen Kontextmodellen verwaltet werden. Somit sind für die Verwaltung der Daten unterschiedliche XML-Dateien zu verwenden. Für den Aufbau und die Struktur der Kontextinformationen sowie für den Zugriff auf die Kontextinformationen hat die getrennte Verwaltung jedoch keine Auswirkungen. Die Kontextinformationen in den verschiedenen Dateien basieren alle auf derselben Datenstruktur und somit auch auf demselben Schema. Für das Bearbeiten der Kontextinformationen muss der Zugriff auf beide Dateien erfolgen.

7.2.2 Verwaltung der Optionsbeziehungen

Zur Verwaltung der Optionsbeziehungen werden in Kapitel 6 Prozessdaten vorgeschlagen. Wie schon im vorhergehenden Abschnitt erwähnt, werden die Prozessdaten in einem BPEL-Prozess im XML-Format gespeichert. Um bei der prototypischen Umsetzung der Provop-Konzepte eine einheitliche Datenverarbeitung zu erreichen, werden auch die Optionsbeziehungen in externen Dateien verwaltet. Dies bringt keinerlei Nachteile mit sich, da zum Einen der WebSphere Integration Developer die Prozessdaten im XML-Format verwaltet und zum Anderen die Definition der Datenstruktur sowie die Zugriffsmethoden schon für die Verwaltung der Kontextinformationen umgesetzt sind.

Für die Definition einer geeigneten Datenstruktur zur Verwaltung der Optionsbeziehungen muss zunächst für jede von Provop unterstützten Beziehungstyp (vgl. Abschnitt 3.4.2) ermittelt werden, welche Angaben erforderlich sind:

- **Implikation.** Wird durch die Angabe der Namen von zwei Optionen spezifiziert, wobei die erstgenannte Option die zweitgenannte impliziert.
- **Wechselseitige Implikation.** Diese Optionsbeziehung kann durch die zweimalige Verwendung der Implikation spezifiziert werden. Dementsprechend wird hierfür keine eigene Datenstruktur definiert.

- **Wechselseitiger Ausschluss.** Wird durch die Angabe der Namen zweier Optionen spezifiziert. Hierbei schließen sich beide Optionen wechselseitig aus.
- **Auswahl n-aus-m.** Wird durch die Angabe eines Vergleichsoperators, einer Zahl n sowie der Menge der m Optionen spezifiziert (vgl. Anforderung 4 in Kapitel 5). Für den Vergleichsoperator seien dabei die Operatoren $=, \neq, >, <, \geq$ sowie $\leq$ erlaubt.

Für die beschriebenen Optionsbeziehungen muss dann ein entsprechendes XML Schema spezifiziert werden. Die zugehörige XML-Datei kann aus beliebig vielen Beziehungen bestehen. Listing 7.3 zeigt eine valide XML-Datei, die aus einer Implikation, einem wechselseitigen Ausschluss sowie einer Auswahl größer 0-aus-3 besteht.

```
<?xml version="1.0" encoding="utf-8" ?>
<optionsbeziehungen xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="optionsbeziehungenSchema.xsd">
  <implikationen>
    <implikation>
      <option1>Option1</option1>
      <option2>Option2</option2>
    </implikation>
  </implikationen>
  <ausschluesse>
    <ausschluss>
      <option1>Option2</option1>
      <option2>Option3</option2>
    </ausschluss>
  </ausschluesse>
  <auswahlenNM>
    <auswahlNM>
      <operator>groesser</operator>
      <vergleichswert>0</vergleichswert>
      <option>Option1</option>
      <option>Option2</option>
      <option>Option3</option>
    </auswahlNM>
  </auswahlenNM>
</optionsbeziehungen>
```

Listing 7.3: Spezifizierte Optionsbeziehungen in einer XML-Datei

Zugriff und Verarbeitung der XML-Datei erfolgt hier ebenfalls über die JDOM-API.

7.2.3 Verwaltung der Optionen

Für die Umsetzung der Verwaltung der Optionen gilt dieselbe Diskussion wie für die Optionsbeziehungen. In Kapitel 6 wird zwar die Verwaltung als Prozessdaten vorgeschlagen, die prototypische Umsetzung erfolgt aber als Verwaltung der Optionen in externen Dateien.

Zur Definition einer geeigneten Datenstruktur für die Optionen sind dabei folgende Attribute notwendig:

- **Name.** Spezifiziert den Namen der Option. Dieser ist in Provop eindeutig und dient zur Identifikation der Optionen.
- **Kontextbedingung.** Spezifiziert in welchem Kontext die Option angewandt werden soll.
- **Status.** Spezifiziert den Status der Option (vgl. Anforderung 7 in Kapitel 5). Zulässige Werte sind `undefined`, `apply` sowie `skip`.
- **Beschreibung.** Dient für eine optionale Beschreibung der Option.

Eine Kontextbedingung muss in einem maschinenlesbaren Format abgelegt werden. Dazu muss überlegt werden, wie eine solche Kontextbedingung aufgebaut werden kann. Listing 7.4 zeigt die Spezifikation der Kontextbedingung in einer Backus-Naur-Form.[3] Dabei ist zu beachten, dass mit dem Nichtterminalsymbol <kontextvariable> ein Name aus der Menge aller vorhandenen Kontextvariablen des Kontextmodells, mit <operator> ein zulässiger Operator sowie mit <wert> ein Wert aus dem Wertebereich der angegebenen Kontextvariable gemeint ist.

```
<expr>      ::= <and> | <or> | <not> | <cond>
<and>       ::= (<expr> AND <expr>)
<or>        ::= (<expr> OR <expr>)
<not>       ::= (NOT <expr>)
<cond>      ::= <kontextvariable> <operator> <wert> | true | false
```

Listing 7.4: Backus-Naur-Form zur Spezifikation der Kontextbedingung

Aus dieser Spezifizierung sowie aus den weiteren Attributen kann nun ein gültiges XML Schema spezifiziert werden. Eine valide XML-Datei der Optionen besteht aus den für das

[3] Die Backus-Naur-Form ist eine Metasprache zur Darstellung kontextfreier Grammatiken [Knu64].

Prozessmodell definierten Optionen. Zugriff und Verarbeitung der XML-Datei erfolgt wie in Abschnitt 7.2.1.

7.2.4 Auswertung der Optionsbeziehungen zur Laufzeit

Wie in Abschnitt 6.2.2 motiviert, soll die Auswertung der Optionsbeziehungen zu Prozessbeginn durch ein prozessinternes Skript erfolgen. Der WebSphere Integration Developer bietet mehrere Alternativen zur Implementierung der Skripte. Diese können entweder in den Exitbedingungen von Aktivitäten oder als eigene Java Snippet Aktivitäten erstellt werden. Zusätzlich gibt es noch die Möglichkeit Programmabschnitte in den Link-Konstrukten eines Allgemeinen Ablaufs zu integrieren.

Um die Auswertung der Optionsbeziehungen transparent zu halten, wird der Auswertungsalgorithmus in einer eigenen Java Snippet Aktivität umgesetzt. Diese Aktivität wird nach dem Start-Ereignis in das Prozessmodell integriert. Abbildung 7.3 zeigt den Ablauf des Algorithmus zur Auswertung der Optionsbeziehungen.

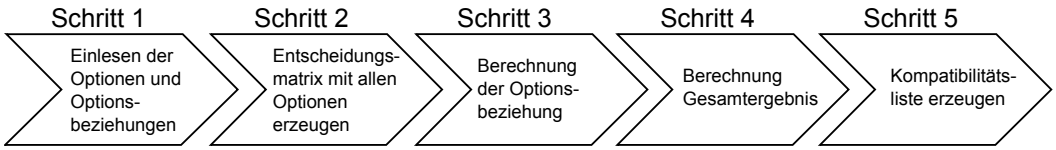

Abbildung 7.3: Ablauf der Auswertung der Optionsbeziehungen

In Schritt 1 werden die XML-Dateien der Optionen sowie der Optionsbeziehungen eingelesen. Daraus wird eine sog. Entscheidungsmatrix zur Auswertung der Optionsbeziehungen erzeugt (vgl. Tabelle 7.1). Diese Entscheidungsmatrix besteht aus einer booleschen Wahrheitstafel mit allen möglichen Wertkombinationen der relevanten statischen und dynamischen Optionen.[4] Zusätzlich besteht die Entscheidungsmatrix aus einer Spalte für jede definierte Optionsbeziehung sowie einer Spalte für das Ergebnis der Auswertung. Nach dem Erzeugen der Entscheidungsmatrix wird jede einzelne Optionsbeziehung gemäß ihrer Definition berechnet. Die Ergebnisse dieser Berechnungen werden anschließend zeilenweise mit einem logischen AND verknüpft und daraus das Ergebnis jeder Zeile berechnet. Entspricht das Ergebnis einer Zeile einer logischen 1, so ist die Anwendung der Optionen hinsichtlich aller Beziehungen kompatibel. Nach dem Berechnen wird die Kompatibilitätsli-

[4] Die Anzahl der möglichen Kombinationen beträgt $2^{Anzahl\ der\ Optionen}$.

ste mit allen zulässigen Kombinationen von Optionen erzeugt. Beispiel 7.1 beschreibt die Auswertung der Optionsbeziehungen sowie die Berechnung der Entscheidungsmatrix.

Beispiel 7.1 (Auswertung der Optionsbeziehungen) Seien O1, O2 und O3 Optionen. Seien folgende Optionsbeziehungen gegeben (vgl. Listing 7.3): *O1 impliziert O2*, *O2 schließt O3 wechselseitig aus* sowie *Auswahl größer 0 aus den Optionen O1, O2 und O3*.

Tabelle 7.1 zeigt die generierte Entscheidungsmatrix. Die ersten drei Spalten zeigen die Kombinationen der drei Optionen. Die Spalten vier, fünf und sechs zeigen die Ergebnisse für die Berechnungen der jeweiligen Optionsbeziehung. Die letzte Spalte zeigt das Ergebnis für die Verknüpfung aller Optionsbeziehungen. Dieses ist 1, falls alle einzelnen Optionsbeziehungen 1 sind. Hier wird ersichtlich, dass es für dieses Beispiel nur drei gültige Kombinationsmöglichkeiten der Optionen gibt. Demnach darf entweder nur Option O2 oder nur Option O3 oder Option O1 mit Option O2 angewandt werden. Andere Kombinationsmöglichkeiten sind nicht erlaubt und würden zu einer Verletzung der Optionsbeziehungen führen.

Tabelle 7.1: Entscheidungsmatrix zur Auswertung der Optionsbeziehungen

O1	O2	O3	O1 -> O2	O2 <-x-> O3	>0 aus O1,O2,O3	Ergebnis
0	0	0	1	1	0	0
0	0	1	1	1	1	**1**
0	1	0	1	1	1	**1**
0	1	1	1	0	1	0
1	0	0	0	1	1	0
1	0	1	0	1	1	0
1	1	0	1	1	1	**1**
1	1	1	1	0	1	0

Die bei der Auswertung der Optionsbeziehungen erzeugte Kompatibilitätsliste hat dabei folgendes Format: In der ersten Zeile der Textdatei stehen die Namen der Optionen getrennt durch ein Semikolon. In den folgenden Zeilen stehen die zulässigen Kombinationen der Optionen. Listing 7.5 zeigt den Inhalt der Kompatibilitätsliste für Beispiel 7.1. Für die drei Optionen O1, O2 und O3 gibt es hierbei drei zulässige Kombinationen (vgl. Tabelle 7.1). Die Wertkombination 1 1 0 steht für die Anwendung von O1 und O2 sowie das Auslassen von O3.

```
O1;O2;O3
001
010
110
```

Listing 7.5: Inhalt der Kompatibilitätsliste nach der Auswertung der Optionsbeziehungen

Die erzeugte Kompatibilitätsliste muss für jede Instanz eines Variantenmodells verwaltet werden. Dabei ist es erforderlich den Pfad der Datei zu speichern, damit während der Ausführung ggf. auf die Datei zugegriffen werden kann. Dementsprechend verwaltet die Prozessinstanz den Pfad in den internen Prozessdaten.

7.2.5 Prüfung der Anwendbarkeit von dynamischen Optionen zur Laufzeit

Ähnlich wie die Auswertung der Optionsbeziehungen soll auch die Prüfung der Anwendbarkeit von dynamischen Optionen durch ein prozessinternes Skript erfolgen (vgl. Abschnitt 6.3.2). Die Anwendbarkeit einer dynamischen Option muss bei Erreichen des ersten OPSplit-Knotens einer Option geprüft werden. Zur Umsetzung im WebSphere Integration Developer wird ein Java Snippet als OPSplit-Knoten in das Prozessmodell integriert. Diese neu eingefügte Aktivität prüft dann die Anwendbarkeit von dynamischen Optionen. Abbildung 7.4 skizziert den Ablauf der Prüfung.

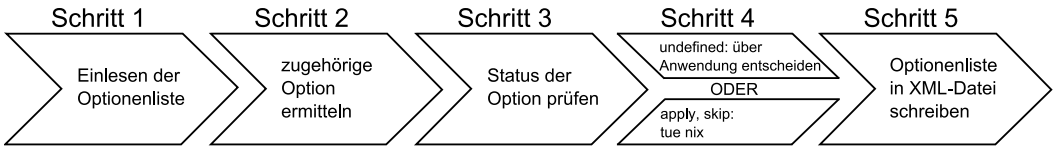

Abbildung 7.4: Ablauf der Anwendbarkeitsprüfung von Optionen

Zunächst wird die XML-Datei der Optionen eingelesen. Daraufhin wird in der Liste der Optionen die zu dem OPSplit-Knoten gehörende Option ermittelt. Die Hinterlegung zu welcher Option ein OPSplit-Knoten gehört, wird für die prototypische Umsetzung durch die Angabe des Namens der Option in der Beschreibung des Knotens erreicht. Sobald die Option ermittelt wurde, wird der Status der Option überprüft. Gemäß Abschnitt 6.3.2 kann der Status die Werte *undefined*, *apply* oder *skip* haben. Ist der Status undefined, wird über die

Anwendung der Option entschieden (vgl. Abbildung 5.2). Nach dem Entscheiden der Anwendbarkeit wird der neue Status der Option in die entsprechende XML-Datei geschrieben. Anschließend wird die Kompatibilitätsliste, welche nach der Auswertung der Optionsbeziehungen erstellt wurde, aktualisiert. Es werden jene Belegungen entfernt, welche nicht mehr mit dem neuen Status der Option übereinstimmen (vgl. Beispiel 7.2). Ist der Status der Option apply oder skip, wurde bereits über ihre Anwendung entschieden.

Beispiel 7.2 (Aktualisierung der Kompatibilitätsliste) Die Kompatibilitätsliste enthält vor der Entscheidung über die Anwendung von Option O1 folgende Werte 001, 010 und 110 (siehe Listing 7.5). Bei der Entscheidung wird festgelegt, dass O1 ausgelassen werden soll. Dementsprechend müssen aus der Kompatibilitätsliste alle Kombinationen entfernt werden, die ein Anwenden von O1 aufweisen (hier: 110). Somit enthält die Kompatibilitätsliste nach der Entscheidung über die Anwendung von O1 nur noch die Kombinationen 001 und 010.

Die eigentliche Anwendung der dynamischen Optionen erfolgt erst an der Kontrollfluss-Kante nach dem OPSplit-Knoten, die das Prozessfragment der dynamischen Option integriert. Dort wird die zugehörige Option bestimmt, der Status der Option gelesen und überprüft. Die Zugehörigkeit der Kante zu einer Option wird wiederum in der Beschreibung hinterlegt. Ist der Status apply, so kommt die dynamische Option zur Ausführung. Ist der Status hingegen skip, so wird die dynamische Option ausgelassen. Abbildung 7.5 zeigt die Prüfung der Anwendbarkeit einer dynamischen Option im WebSphere Integration Developer. Im OPSplit-Knoten erfolgt die Entscheidung über die Anwendung der Option, während die Kante den im OPSplit-Knoten gesetzten Status der Option überprüft.

Die in diesem Abschnitt beschriebene Prüfung der Anwendbarkeit einer dynamischen Option findet an jedem OPSplit-Knoten im Prozessmodell statt.

7.2.6 Korrekte Synchronisation

Wie in Abschnitt 6.3.2 motiviert, soll die korrekte Synchronisation der Ausführungspfade an einem OPJoin-Knoten mit Hilfe einer speziellen Eingangsbedingung erfolgen. Zur Auswertung der Eingangsbedingung müssen die Belegungen der eingehenden Kanten ermittelt und entsprechend der betreffende Ausführungspfad über die ausgehende Kante des OPJoin-Knotens ausgeführt oder ausgelassen werden. Die prototypische Umsetzung im WebSphere Integration Developer kann nicht direkt erfolgen, weil nicht auf die Belegungen der Kanten in einem Allgemeinen Ablauf zugegriffen werden kann, sondern nur auf die

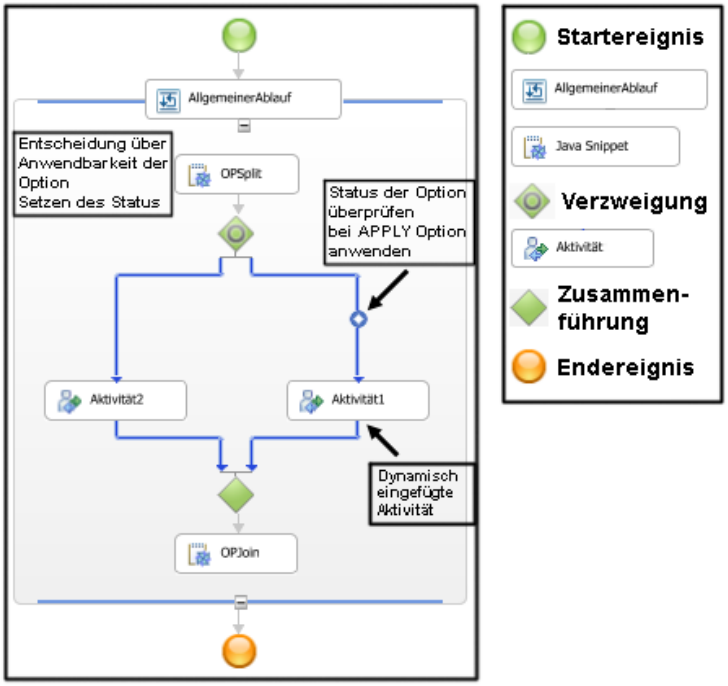

Abbildung 7.5: Prüfung der Anwendbarkeit einer dynamischen Option

Ausführungszustände von bereits ausgeführten Aktivitäten. Stattdessen ist ein Workaround erforderlich.

Dementsprechend muss die Eingangsbedingung auf die Zustände der vorherigen Aktivitäten zurückgreifen. Der Zustand `finished` besagt, dass die Aktivität erfolgreich beendet wurde, der Zustand `skipped` besagt, dass die Aktivität ausgelassen wurde.[5] Zur Spezifizierung der Eingangsbedingung ist Aktivität 2 in Abbildung 7.5 jene Aktivität, die sich auf dem Pfad der regulären Kontrollflusskante befindet und Aktivität 1 jene Aktivität, die sich auf dem Pfad der Operationskante befindet. Die Eingangsbedingung evaluiert zu `true`, falls Aktivität 2 den Zustand `finished` und Aktivität 1 den Zustand `finished` oder `skipped` hat. Zur prototypischen Umsetzung des OPJoins müssen die Namen der zu evaluierenden Aktivitäten in der Beschreibung des Knotens hinterlegt werden, damit ihre Zustände ermittelt werden können. Listing 7.6 zeigt den Java-Code zur Überprüfung der Eingangsbedingung an einem OPJoin-Knoten.

[5] Im WebSphere Integration Developer gibt es noch eine Vielzahl weiterer Zustände für die Aktivitäten, zur Abbildung der Eingangsbedingung werden aber nur die beiden benötigt.

```
//Informationen bzgl. der Aktivitäten beschaffen
ActivityInstanceData regAid = activityInstance(regAktivityName);
ActivityInstanceData opeAid = activityInstance(opeAktivityName);
//Überprüfen der Eingangsbedingung
if(regAid.getExecutionState() == ActivityInstanceData.STATE_FINISHED &&
 (opeAid.getExecutionState() == ActivityInstanceData.STATE_FINISHED ||
  opeAid.getExecutionState() == ActivityInstanceData.STATE_SKIPPED)) {
  //Eingangsbedingung evaluiert zu true
  [..]
}
[..]
```

Listing 7.6: Überprüfung der Eingangsbedingung am OPJoin-Knoten

Nachdem die Eingangsbedingung evaluiert wurde, muss entsprechend des Resultats der Prüfung die weitere Ausführung gesteuert werden. Evaluiert die Eingangsbedingung zu `false`, so müssen alle Aktivitäten auf dem Pfad der ausgehenden Kante ausgelassen werden. Um dies zu erreichen, muss zunächst geklärt werden, wie der Kontrollfluss im Allgemeinen Ablauf gesteuert wird. Dort arbeitet der WebSphere Integration Developer mit sog. *Live Token*, d.h. werden bspw. an einem verzweigenden Gateway zwei Token erzeugt, so erwartet das zusammenführende Gateway wiederum zwei Token. Dies führt im Zusammenhang mit der speziellen Eingangsbedingung zu einem Problem:

Bei einer negativen Evaluation der Eingangsbedingung des OPJoins soll der Pfad ausgelassen werden bzw. das für den Pfad gültige Token soll gelöscht werden. Somit wartet jedoch das zusammenführende Gateway auf ein nicht ankommendes Token und der Prozess befindet sich in einer Verklemmungssituation.

Um die Verklemmung zu lösen, wird zur prototypischen Umsetzung eine zusätzliche Kante um das auszulassende Prozessfragment modelliert. Auf diese Weise gelangt auch bei einer negativen Evaluation der Eingangsbedingung ein Token über die zusätzliche Kante zum zusammenführenden Gateway. Somit wird ein korrekter Kontrollfluss gewährleistet. Allerdings muss dem Modellierer bekannt sein, welches Prozessfragment bei einer negativen Evaluation der Eingangsbedingung ausgelassen und mit einer zusätzlichen Kante umgangen werden soll.

Abbildung 7.6 zeigt, wie der Workaround zur prototypischen Umsetzung der korrekten Synchronisation realisiert wird. Zur Strukturierung wird eine sog. *leere Aktion* eingefügt, der den Ausführungspfad der Option mit dem regulären Kontrollfluss zusammenführt. An der

ausgehenden Kante des OPJoin-Knotens erfolgt die Evaluation der Eingangsbedingung. Falls diese zu `false` evaluiert, wird die Ausführung über die zusätzlich eingefügte Kante fortgesetzt.

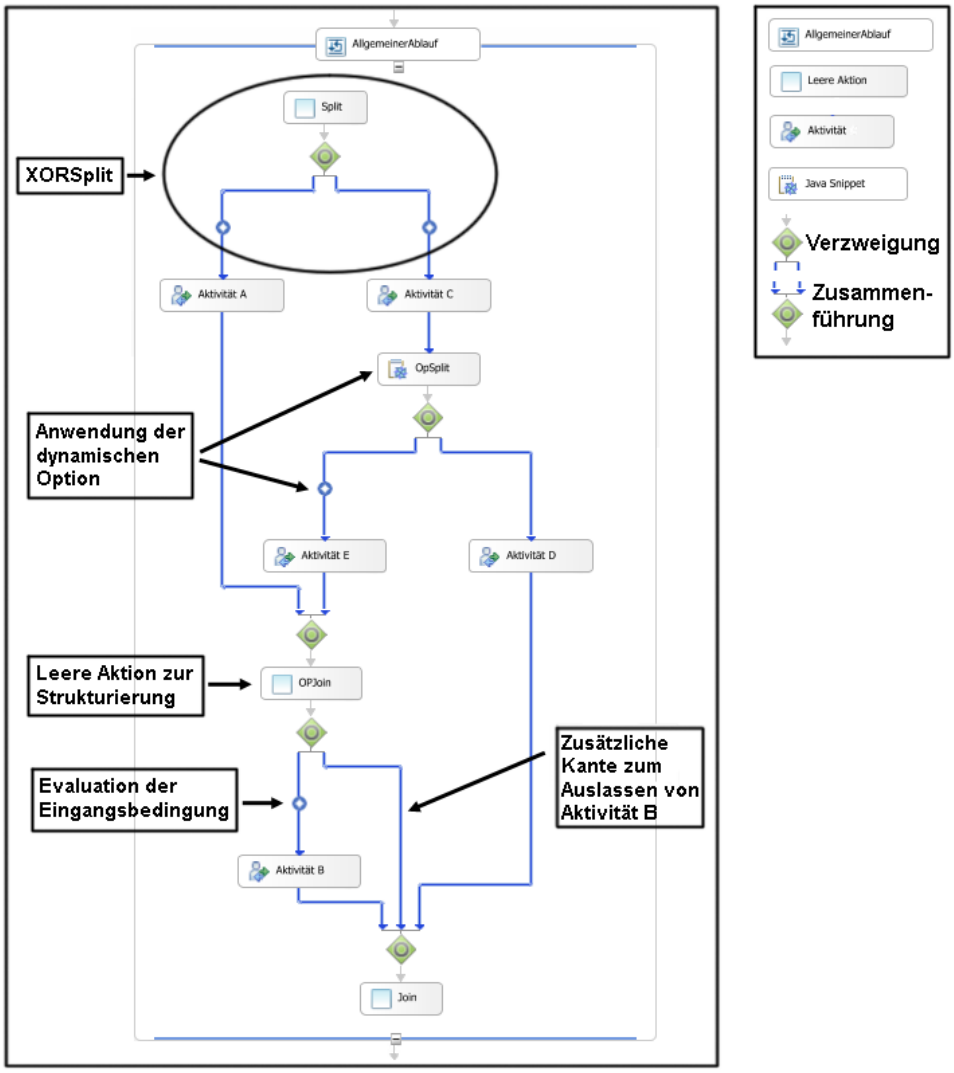

Abbildung 7.6: Workaround zur Umsetzung der korrekten Synchronisation

Der gewählte Workaround zur Umsetzung der korrekten Synchronisation ist nicht optimal, da der Modellierer beim Erstellen des Prozessmodells wissen muss, zwischen welchen Elementen die zusätzliche Kante eingefügt werden muss. Die Problematik kommt durch die Steuerung des Kontrollflusses über Tokens. Würde im Allgemeinen Ablauf stattdessen eine

Dead Path Elimination[6] zur Anwendung kommen, so würde bei einer negativen Evaluation der Eingangsbedingung der restliche Ausführungspfad bis zum zusammenführenden Gateway ausgelassen. Dadurch entfällt das Modellieren der zusätzlichen Kante.

7.2.7 Zusammenfassung des Prototypen

In diesem Abschnitt werden alle Schritte aufgelistet, die zur prototypischen Umsetzung des Provop-Lösungsansatzes im WebSphere Integration Developer beachtet werden müssen (vgl. Abbildung 7.7).

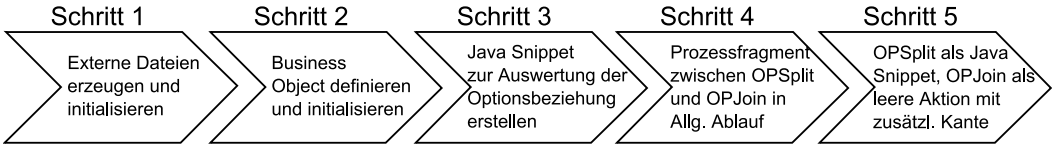

Abbildung 7.7: Ablauf zur prototypischen Umsetzung des Provop-Lösungsansatzes

Zu Beginn müssen die externen Dateien zur Verwaltung der Kontextinformationen, der Optionsbeziehungen sowie der Optionen erzeugt und mit Daten gefüllt werden. Hierbei ist zu beachten, dass für jede Prozessinstanz jeweils eine Datei mit den lokalen Kontextinformationen, den für die Prozessinstanz gültigen Optionsbeziehungen sowie den für die Prozessinstanz definierten Optionen erstellt werden muss. Zusätzlich muss eine Datei für die globalen Kontextinformationen erzeugt werden. Diese Datei ist dann für alle Instanzen einer Prozessfamilie gültig. Für jede Prozessinstanz muss ein Business Object angelegt werden, welches die Pfade zu den externen Dateien verwaltet.

Dem Prozessmodell muss unmittelbar nach dem Startereignis des Prozesses ein Java Snippet hinzugefügt werden. Dort findet die Auswertung der Optionsbeziehungen statt (vgl. Abschnitt 7.2.4). Bei der Auswertung wird die Kompatibilitätsliste mit den zulässigen Kombinationen von Optionen erzeugt. Hierbei ist zu beachten, dass auch der Pfad dieser Datei in dem vorhin beschriebenen Business Object verwaltet wird, da auf die Kompatibilitätsliste bei der Prüfung der Anwendbarkeit von Optionen zur Laufzeit zugegriffen werden muss.

Zum dynamischen Anwenden von Optionen müssen die Kontrollfluss-Konstrukte der Änderungsoperationen mit den entsprechenden OPSPlit und OPJoin-Knoten in einen Allge-

[6] Unter der Dead Path Elimination versteht man das Entfernen von ganzen Pfaden bei der Ausführung von Prozessen. Wird beispielsweise die Kante einer parallelen Verzweigung abgewählt, so wird der gesamte Pfad abgewählt [LR00].

meinen Ablauf modelliert werden. Der OPSplit-Knoten wird hierbei durch ein Java Snippet realisiert. Dort findet die Entscheidung über die Anwendbarkeit von dynamischen Optionen statt (vgl. Abschnitt 7.2.5). Dabei wird geprüft, ob die zugehörige dynamische Option im aktuellen Kontext und hinsichtlich ihrer Kompatibilität mit anderen Optionen angewandt werden darf oder nicht (vgl. Abbildung 7.5).

Der OPJoin wird als leere Aktion zur Strukturierung in das Prozessmodell eingefügt. Die ausgehende Kante überprüft die Eingangsbedingung. Zusätzlich wird eine leere Kontrollflusskante in das Prozessmodell eingefügt (vgl. Abschnitt 7.2.6).

7.3 Architektur des Prototypen

In diesem Abschnitt wird die Gesamtarchitektur des entwickelten Prototypen vorgestellt. Abbildung 7.8 zeigt die einzelnen Komponenten des Prototypen zur Umsetzung des Provop-Lösungsansatzes.

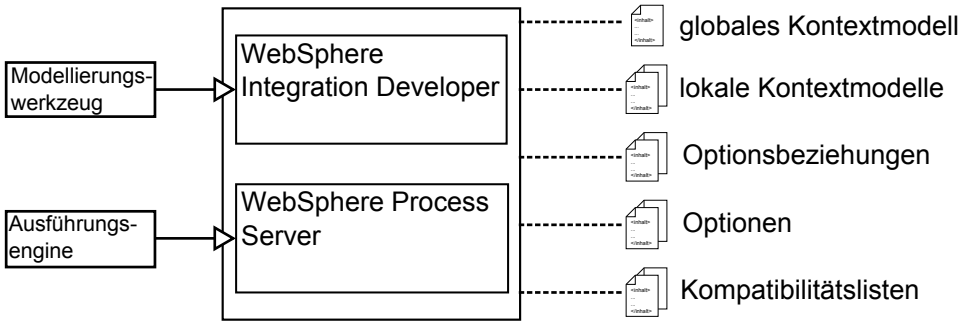

Abbildung 7.8: Architektur des Prototypen

Die Modellierung der Prozessmodelle erfolgt im WebSphere Integration Developer. Die Provop-Konzepte werden dabei wie in Abschnitt 7.2 beschrieben abgebildet. Die explizite Ausführung der einzelnen Prozessinstanzen erfolgt auf dem WebSphere Process Server. Während der Ausführung wird auf die verschiedenen externen Dateien zugegriffen, in denen Informationen gespeichert sind (vgl. Abschnitt 7.2).

7.4 Fallbeispiel

Nachdem im vorhergehenden Abschnitt der Prototyp vorgestellt wurde, wird in diesem Abschnitt der Einsatz des Provop-Lösungsansatzes anhand eines Beispiels vorgestellt.

Problembeschreibung. Es sei der in Abbildung 2.1 dargestellte Prozess als Basisprozess für die Domäne Werkstatt gegeben. Das Kontextmodell besteht aus den Kontextvariablen `Auslastung` und `Kostenvoranschlag`. Die Auslastung kann die Werte `niedrig`, `mittel` sowie `hoch` annehmen und hat einen globalen Geltungsbereich. Sie kann sich während der Ausführung ändern und wird demnach als dynamisch spezifiziert. Der Kostenvoranschlag kann Werte aus den Intervallen $>$ `800 €` und $\leq$ `800 €` annehmen. Er hat einen lokalen Geltungsbereich, wird erst zur Laufzeit festgelegt und als dynamisch spezifiziert.

Zur Ableitung einer Prozessvariante sind folgende Optionen definiert:

- **Option 1.** Zur Sicherstellung einer hohen Kundenzufriedenheit werden bei einer niedrigen Auslastung der Werkstatt eine zusätzliche Wartung nach der Annahme des Fahrzeugs sowie eine abschließende Prüfung nach der Reparatur in das Prozessmodell eingefügt. Die zugehörige Kontextbedingung lautet `Auslastung = niedrig`.
- **Option 2.** Ist der Kostenvoranschlag, welcher bei der Diagnose festgelegt wird, mit $>$ 800 € gegeben, so wird nach der Diagnose beim Kunden nachgefragt, ob er die Reparatur durchführen lassen möchte. Es wird die Aktivität Rückfrage eingefügt. Die Kontextbedingung lautet `Kostenvoranschlag > 800 €`.
- **Option 3.** Um bei einer hohen Auslastung der Werkstatt möglichst viele Aufträge bearbeiten zu können ohne die Kunden durch lange Reparaturzeiten zu verärgern, wird die Reparatur ggf. in einer Partner-Werkstatt erledigt. Hierzu wird die Rolle des Bearbeiters der Reparatur geändert und zusätzlich ein Transport zur und eine Abholung von der externen Werkstatt in das Prozessmodell eingefügt. Die Kontextbedingung lautet `Auslastung = hoch`.

Für die Optionen ist die Optionsbeziehung definiert, dass bei einem Kostenvoranschlag von $>$ 800€ das Fahrzeug nicht von einer anderen Werkstatt repariert werden darf, da solche Aufträge für die eigene Werkstatt wirtschaftlicher sind. Somit schließen sich Option 2 und Option 3 wechselseitig aus.

Abbildung 7.9 zeigt die Provop-Darstellung für das Fallbeispiel mit Basisprozess (a), den drei Optionen (b) sowie der definierten Optionsbeziehung (c).

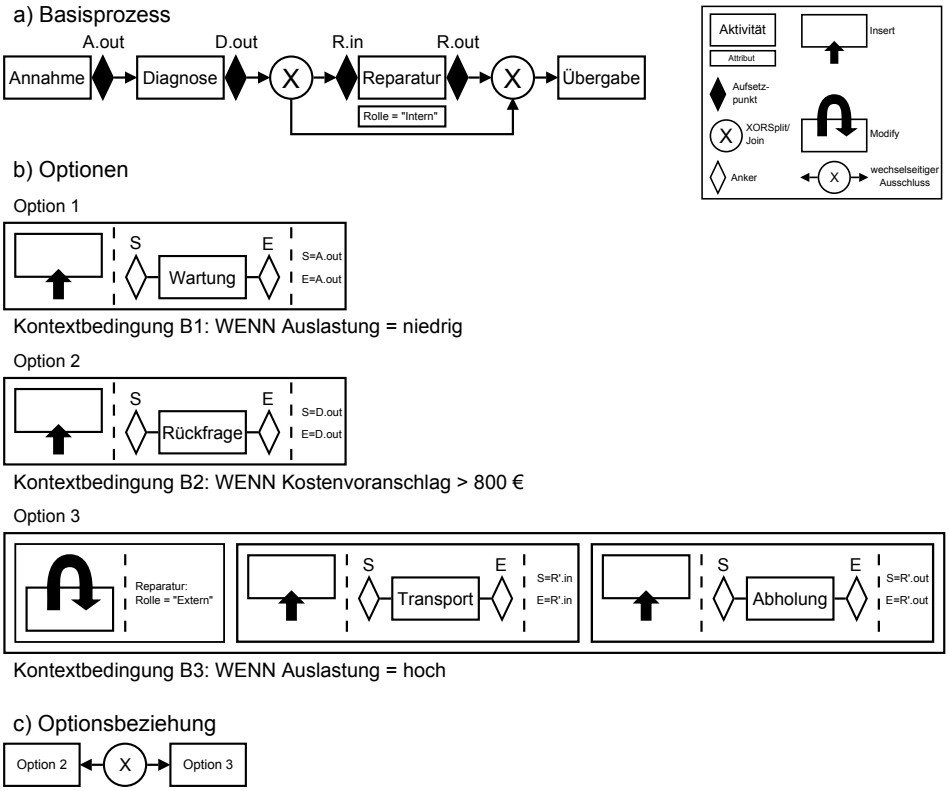

Abbildung 7.9: a) Basisprozess mit b) Optionen und c) Optionsbeziehung

Ablauf des Prozesses. Nach der Instanziierung des Prozessmodells erfolgt die Auswertung der definierten Optionsbeziehungen. Dabei ergeben sich für die drei Optionen folgende zulässige Kombinationen (vgl. Abbildung 7.10):[7]

- **000.** Keine der drei Optionen wird angewandt.
- **001.** Nur Option 3 wird angewandt.
- **010.** Nur Option 2 wird angewandt.
- **100.** Nur Option 1 wird angewandt.
- **101.** Option 1 wird gemeinsam mit Option 3 angewandt.

[7] Eine gemeinsame Anwendung von Option 2 und Option 3 wird aufgrund der definierten Optionsbeziehung ausgeschlossen.

- **110.** Option 1 wird gemeinsam mit Option 2 angewandt.

Nach der Annahme des Fahrzeugs wird der erste OPSplit-Knoten von Option 1 erreicht. An dieser Stelle wird zunächst der Status der Option geprüft. Da dieser `undefined` ist, d.h. über die Anwendung dieser Option noch nicht entschieden wurde, wird die Kontextbedingung der Option geprüft. Unter der Annahme, dass der aktuelle Wert der dynamischen Kontextvariable `niedrig` ist, ist die Kontextbedingung der Option erfüllt. Anschließend wird mit Hilfe der Kompatibilitätsliste ermittelt, ob eine Anwendung von Option 1 kompatibel ist. Dies ist im Beispiel der Fall. Dementsprechend darf die Option angewandt werden. Der Status der Option wird auf `apply` gesetzt und die Kompatibilitätsliste aktualisiert. Diese enthält dann folgende Kombinationen (vgl. Abbildung 7.10):

- **100.** Nur Option 1 wird angewandt.
- **101.** Option 1 wird gemeinsam mit Option 3 angewandt.
- **110.** Option 1 wird gemeinsam mit Option 2 angewandt.

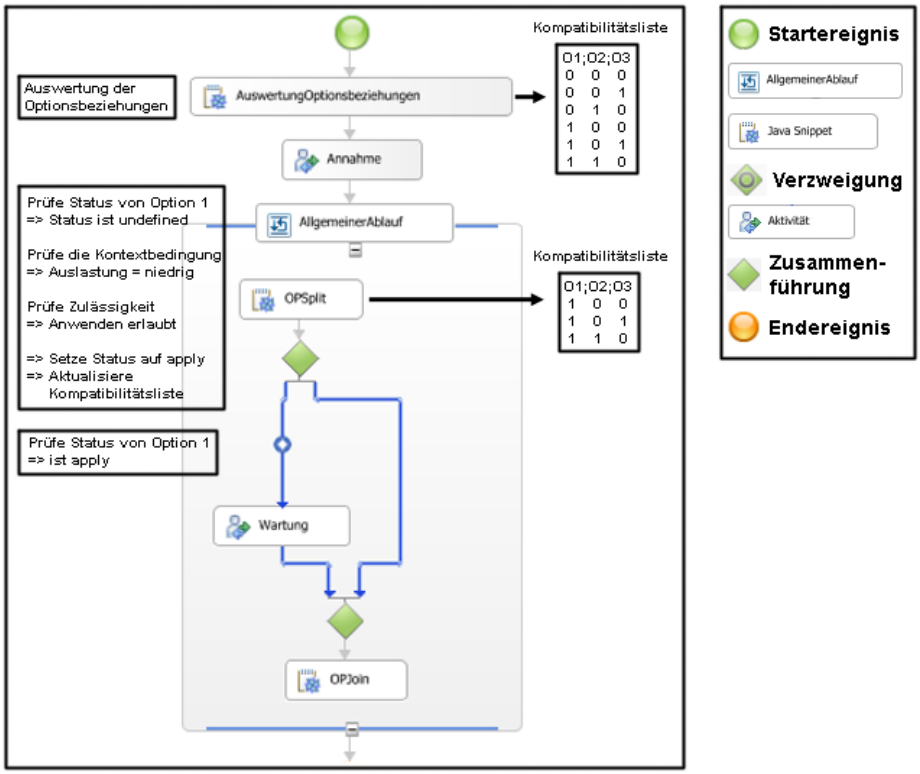

Abbildung 7.10: Ausschnitt des Beispielszenarios

Da Option 1 angewandt wird, erfolgt eine Wartung des Fahrzeugs. Nach der Wartung wird der Schaden diagnostiziert. Dabei ergibt sich ein Kostenvoranschlag von 900 €. Anschließend wird der erste OPSplit-Knoten von Option 2 erreicht. Auch hier wird zunächst der Status (hier `undefined`) und daraufhin die Kontextbedingung geprüft. Diese ist gültig, da der Kostenvoranschlag $>$ 800 € ist. Nun wird mit Hilfe der Kompatibilitätsliste geprüft, ob eine Anwendung von Option 2 kompatibel ist. Dies ist der Fall und Option 2 darf angewandt werden. Der Status der Option wird auf `apply` gesetzt und die Kompatibilitätsliste aktualisiert. Diese enthält dann folgende Kombinationen:

- **110.** Option 1 wird gemeinsam mit Option 2 angewandt.

Da Option 2 angewandt werden darf, wird beim Kunden nachgefragt, ob er die Reparatur in der Werkstatt erledigen lassen will. Dieser bestätigt die Reparatur. Daraufhin ändert sich die Auslastung der Werkstatt. Diese ist nun hoch. Nun wird der erste OPSplit-Knoten von Option 3 erreicht. Es wird zunächst der Status (hier `undefined`) und daraufhin die Kontextbedingung geprüft. Diese ist gültig, da die Auslastung der Werkstatt aktuell mit `hoch` gegeben ist. Nun wird in der Kompatibilitätsliste ermittelt, ob eine Anwendung von Option 3 zulässig ist. Dies ist nicht der Fall, da Option 2 bereits angewandt und aufgrund des wechselseitigen Ausschlusses die Anwendung von Option 3 ausgeschlossen ist (siehe Kompatibilitätsliste). Der Status der Option wird auf `skip` gesetzt und die Option nicht angewandt.

Nach der Reparatur des Fahrzeugs in der Werkstatt wird der zweite OPSplit-Knoten von Option 1 erreicht. Dort wird der Status der Option geprüft. Da dieser mit `apply` gegeben ist, wird die zweite Änderungsoperation von Option 1 ebenfalls angewandt. Hierbei ist zu beachten, dass die Anwendung erfolgt, obwohl die Kontextbedingung nicht mehr gültig ist. Dies liegt daran, dass die Atomarität der Option auch zur Laufzeit gewährleistet wird. Nach der eingefügten Prüfung des Fahrzeugs erfolgt die Übergabe an den Kunden und die Prozessinstanz terminiert.

7.5 Zusammenfassung

In diesem Kapitel wird die prototypische Umsetzung des Provop-Lösungsansatzes vorgestellt. Als Zielplattform für die Umsetzung wird dazu der WebSphere Integration Developer gewählt.

Die Verwaltung der Kontextinformationen, der Optionsbeziehungen sowie der Optionen erfolgt in externen XML-Dateien. Zu allen drei Punkten wird die Datenstruktur zur Speicherung in einer XML-Datei sowie der Zugriff auf die Daten erörtert.

Die Auswertung der Optionsbeziehungen erfolgt in einem Java Snippet nach Instanziierung des Prozessmodells. Dort wird eine Entscheidungsmatrix zur Bestimmung der zulässigen Kombinationen aufgebaut und berechnet. Anschließend werden diese dann in einer Kompatibilitätsliste gespeichert.

Die Prüfung der Anwendbarkeit von dynamischen Optionen erfolgt ebenfalls in einem Java Snippet. Dort wird gemäß Abbildung 5.2 über die Anwendbarkeit einer Option entschieden und der Status der Option entsprechend gesetzt.

Die korrekte Synchronisation am OPJoin-Knoten wird durch einen Workaround erreicht. Die Umsetzung der korrekten Synchronisation erfolgt durch das Einfügen einer leeren Aktion in das Prozessmodell. Zudem wird in der ausgehenden Kante die Eingangsbedingung durch Auswertung der Ausführungszustände der Aktivitäten evaluiert. Das etwaige Auszulassen des Prozessfragments erfolgt über eine zusätzlich eingefügte Kante.

Die Praxistauglichkeit der prototypischen Umsetzung wird anhand eines Fallbeispiels vorgestellt. Dadurch wird gezeigt, dass der Provop-Lösungsansatz zur Ausführung von Prozessvarianten im WebSphere Integration Developer abgebildet werden kann. Nur für die Abbildung der korrekten Synchronisation am OpJoin-Knoten erfolgt die Umsetzung mit Hilfe eines Workarounds. Alle anderen Provop-Konzepte können komfortabel umgesetzt werden. Der entwickelte Prototyp kann im Bereich des Variantenmanagements eingesetzt werden und stellt eine Unterstützung für den Anwender bei der Ausführung von Prozessvarianten dar.

8 Zusammenfassung und Ausblick

Das Management von Prozessvarianten wird heutzutage nur unzureichend von aktuellen Prozessmanagement-Systemen unterstützt. Dabei werden Prozessvarianten oftmals nur in separaten Modellen oder alle Varianten eines Prozesses in einem Prozessmodell abgebildet. Dies bringt eine Reihe von Nachteilen mit sich. Dementsprechend wird bei der Daimler AG im Rahmen des Forschungsprojekts Provop ein generischer Lösungsansatz für ein transparentes und kontextabhängiges Management von Prozessvarianten über alle Phasen des Prozesslebenszyklus entwickelt.

Die Grundidee von Provop basiert darauf, anhand eines Basisprozesses und zugehörigen variantenspezifischen Abweichungen, alle relevanten Prozessvarianten ableiten zu können. Zur Spezifizierung der Abweichungen bietet Provop verschiedene höherwertige Änderungsoperationen an, wie etwa das Einfügen von Prozessfragmenten. Diese Änderungsoperationen können zu sog. Optionen gruppiert werden.

Zur Konfiguration einer Prozessvariante werden Optionen ausgewählt, welche auf den Basisprozess angewandt werden, um daraus die Prozessvariante zu generieren. Diese Auswahl hängt zumeist von speziellen Rahmenbedingungen ab, d.h. vom variantenspezifischen Kontext. In Provop wird der Anwender bei dieser kontextbasierten Konfiguration unterstützt, indem Kontextbedingungen für Optionen spezifiziert werden können. Der Kontext wird in einem Kontextmodell bestehend aus verschiedenen Kontextvariablen verwaltet. Zudem können zur Konfiguration Beziehungen zwischen den Optionen definiert werden, um so strukturelle und semantische Abhängigkeiten zwischen den Optionen zu spezifizieren. Des Weiteren gibt es in Provop sog. dynamische Kontextvariablen. Dadurch ist es möglich, Optionen dynamisch zur Laufzeit auf den Basisprozess anzuwenden.

Im Rahmen dieser Diplomarbeit wird untersucht, wie das anhand der Provop-Konzepte fachlich modellierte und konfigurierte Prozessmodell in ein technisches Workflow-Modell zur Ausführung in einem WfMS überführt werden kann. Dabei wird auf die Übertragung der relevanten Provop-Konstrukte fokussiert.

Hierzu werden zunächst verwandte Arbeiten diskutiert, die sich mit dem Management von Prozessvarianten beschäftigen. Es wird ein Überblick über aktuelle Forschungsansätze sowie heutige WfMS gegeben. Dabei wird ersichtlich, dass Prozessvarianten bei der Ausführung nicht ausreichend unterstützt werden.

Im nächsten Schritt werden die Anforderungen an die Ausführung von Prozessvarianten durch den Provop-Lösungsansatz diskutiert. Dabei wird erläutert, was zur Übertragung der Provop-Konzepte in ein technisches Workflow-Modell zu beachten ist. Es werden die Anforderungen beschrieben, die sich durch die Verwaltung des Kontexts ergeben. Zudem wird erörtert, wie die strukturellen und semantischen Abhängigkeiten während der Ausführung gewährleistet werden können. Zusätzlich werden die Anforderungen der Provop-spezifischen Kontrollfluss-Konstrukte für die Ausführung von Prozessvarianten erklärt.

Daraufhin werden für die vorgestellten Anforderungen verschiedene Abbildungsmöglichkeiten diskutiert. Dabei werden zu jeder Anforderung alternative Lösungsansätze zur Abbildung beschrieben, Vor- und Nachteile diskutiert sowie eine Präferenz für die jeweilige Abbildung angegeben. Hierbei wird darauf geachtet, dass die Abbildungsmöglichkeiten möglichst kompatibel miteinander sind und dadurch ein durchgängiges Konzept der Abbildung verfolgt werden kann. Für die gewählten Abbildungsmöglichkeiten wird anschließend diskutiert, ob sie von kommerziellen WfMS unterstützt werden und somit für einen Einsatz des Provop-Lösungsansatzes in der Praxis realisierbar sind. Dabei wird ersichtlich, dass sich der Provop-Lösungsansatz zur Ausführung von Prozessvarianten anhand der gewählten Abbildungsmöglichkeiten prinzipiell in allen betrachteten Systemen umsetzen lässt.

Zum Abschluss der Arbeit wird die prototypische Umsetzung des Provop-Lösungsansatzes im WebSphere Integration Developer von IBM vorgestellt. Zunächst wird die Zielplattform beschrieben. Daraufhin werden die konkreten Implementierungen der einzelnen Abbildungsmöglichkeiten für das System detailliert erläutert und die Gesamtarchitektur zur Umsetzung des Provop-Lösungsansatzes erörtert. Die Implementierung wird anhand eines Fallbeispiels auf ihre Anwendbarkeit geprüft. Dabei werden einige besondere Aspekte der Ausführung einer Prozessvariante behandelt, die speziell von Provop adressiert werden.

Die vorliegende Arbeit zeigt, dass der Provop-Lösungsansatz zur Ausführung von Prozessvarianten in einem kommerziellen WfMS umgesetzt werden kann. Sie stellt somit die Grundlage für weitere Implementierungen des Provop-Lösungsansatzes dar. Der Großteil der Provop-Konzepte konnte anhand der gewählten Abbildungsmöglichkeiten komfortabel umgesetzt werden. Manche Konzepte konnten allerdings nur mittels eines Workarounds

realisiert werden, wie bspw. die korrekte Synchronisation der Ausführungspfade nach der Integration eines Prozessfragments. Hierzu bedarf es noch weiterer Forschungs- und Entwicklungsarbeit. Dazu ist zu sagen, dass ein in dieser Arbeit betrachtetes WfMS für dieses Konzept eine geeignete proprietäre Lösung anbietet. Demnach wäre eine Umsetzung des Provop-Lösungsansatzes in ADEPT eine interessante Weiterführung der vorliegenden Arbeit.

Literaturverzeichnis

[AADH04] W.M.P. VAN DER AALST, L. ALDRED, M. DUMAS, A.H.M. TER HOFSTEDE: Design and Implementation of the YAWL System. In: A. PERSSON (Hrsg.), J. STIRNA (Hrsg.): *CAiSE*, Springer, 2004 (LNCS 3084), S. 142–159

[ADG+08] W.M.P. VAN DER AALST, M. DUMAS, F. GOTTSCHALK, A.H.M. TER HOFSTEDE, M. LA ROSA, J. MENDLING: Correctness-Preserving Configuration of Business Process Models. In: J. FIADEIRO (Hrsg.), P. INVERARDI (Hrsg.): *FASE*, Springer, 2008 (LNCS 4961), S. 46–61

[AH05] W.M.P. VAN DER AALST, A.H.M. TER HOFSTEDE: YAWL: yet another workflow language. In: *Information Systems* 30 (2005), Juni, Nr. 4, S. 245–275

[AHAE07] M. ADAMS, A.H.M. TER HOFSTEDE, W.M.P. VAN DER AALST, D. EDMOND: Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In: *15th International Conference on Cooperative Information Systems (CoopIS)*, Springer, 2007 (LNCS 4803), S. 95–112

[AHEA06] M. ADAMS, A.H.M. TER HOFSTEDE, D. EDMOND, W.M.P. VAN DER AALST: Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In: *OTM Conferences (1)*. 2006, S. 291–308

[AHKB03] W.M.P. VAN DER AALST, A.H.M. TER HOFSTEDE, B. KIEPUSZEWSKI, A. BARROS: Workflow Patterns. In: *Distributed and Parallel Databases* 14 (2003), Juli, Nr. 1, S. 5–51

[BC06] G. BEERS, J. CAREY: WebSphere Process Server Business State Machines concepts and capabilities, Part 1: Exploring basic concepts / IBM. 2006. – Technischer Bericht

[BPE07] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS: Business Process Execution Language Version 2.0. 2007. – Technischer Bericht

[BPM09] OBJECT MANAGEMENT GROUP: Business Process Modeling Notation Version 1.2. Object Management Group (OMG), Januar 2009. – Technischer Bericht

[Dad96] P. DADAM: *Verteilte Datenbanken und Client/Server-Systeme*. Springer, 1996

[DRRM+09] P. DADAM, M. REICHERT, S. RINDERLE-MA, K. GOESER, U. KREHER, M. JURISCH: Von ADEPT zur AristaFlow BPM Suite - Eine Vision wird Realität: Correctness by Construction und flexible, robuste Ausführung von Unternehmensprozessen. In: *EMISA Forum* 29 (2009), Januar, Nr. 1, S. 9–28

[GAJVR08] F. GOTTSCHALK, W.M.P. VAN DER AALST, M. JANSEN-VULLERS, M. LA ROSA: Configurable Workflow Models. In: *International Journal of Cooperative Information Systems* 17 (2008), Nr. 2, S. 177–221

[Hal10] A. HALLERBACH: *Durchgängiges und kontextsensitves Management von Prozessvarianten*, Universität Ulm, Dissertation, 2010. – erscheint demnächst

[HBR08a] A. HALLERBACH, T. BAUER, M. REICHERT: Anforderungen an die Modellierung und Ausführung von Prozessvarianten. In: *Datenbank Spektrum* 24 (2008), Februar, S. 48–58

[HBR08b] A. HALLERBACH, T. BAUER, M. REICHERT: Modellierung und Darstellung von Prozessvarianten in Provop. In: *Modellierung'08 Conference*, Koellen-Verlag, März 2008 (LNI P-127), S. 41–56

[HBR09a] A. HALLERBACH, T. BAUER, M. REICHERT: Configuration and Management of Process Variants. In: M. ROSEMANN (Hrsg.), J. V. BROCKE (Hrsg.): *Handbook on Business Process Management*, Springer, 2009. – (forthcoming)

[HBR09b] A. HALLERBACH, T. BAUER, M. REICHERT: Correct Configuration of Process Variants in Provop / University of Ulm. Ulm : University of Ulm, Faculty of Electrical Engineering and Computer Science, Februar 2009 (UIB-2009-03). – Technical Report

[HBR09c] A. HALLERBACH, T. BAUER, M. REICHERT: Guaranteeing Soundness of Configurable Process Variants in Provop. In: *11th IEEE Conference on Commerce and Enterprise Computing (CEC'09)*, IEEE Computer Society Press, Juli 2009

[IBM09] IBM: *IBM WebSphere Process Server 6.2 - IBM WebSphere Integration Developer 6.2*, 2009. – Technischer Bericht

[KEP00] G. KNOLMAYER, R. ENDL, M. PFAHRER: Modeling Processes and Workflows by Business Rules. In: W.M.P. VAN DER AALST (Hrsg.), J. DESEL (Hrsg.), A. OBERWEIS (Hrsg.): *Business Process Management*, Springer, 2000 (LNCS 1806), S. 16–29

[KNS92] G. KELLER, M. NÜTTGENS, A.W. SCHEER: *Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK)*. Institut für Wirtschaftsinformatik, Heft 89, Saarbrücken, Deutschland, 1992

[Knu64] D.E. KNUTH: Backus Normal Form vs. Backus Naur Form. In: *Communications of the ACM* 7 (1964), Nr. 12, S. 735–736

[LR00] F. LEYMANN, D. ROLLER: *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000

[PSSA07] M. PESIC, M. SCHONENBERG, N. SIDOROVA, W.M.P. VAN DER AALST: Constraint-Based Workflow Models: Change Made Easy. In: R. MEERSMAN (Hrsg.), Z. TARI (Hrsg.): *OTM Conferences (1)*, Springer, 2007 (LNCS 4803), S. 77–94

[PSWW05] F. PUHLMANN, A. SCHNIEDERS, J. WEILAND, M. WESKE: PESOA - Variability Mechanisms for Process Models / Hasso-Plattner-Institut, Potsdam. 2005 (17/2005). – Technischer Bericht

[RA07] M. ROSEMANN, W.M.P. VAN DER AALST: A configurable reference modelling language. In: *Information Systems* 32 (2007), Nr. 1, S. 1–23

[Rei00] M. REICHERT: *Dynamische Ablaufänderungen in Workflow-Management-Systemen*, Universität Ulm, Dissertation, 2000

[RHAE04] N. RUSSELL, A.H.M. TER HOFSTEDE, W.M.P. VAN DER AALST, D. EDMOND: Workflow Data Patterns (Revised Version) / BPMcenter.org. 2004 (BPM Center Report BPM-04-01 BPM-04-01). – Technischer Bericht

[RMT09] H. REIJERS, R. MANS, R. VAN DER TOORN: Improved model management with aggregated business process models. In: *Data & Knowledge Engineering* 68 (2009), Nr. 2, S. 221–243

[RRMD09] M. REICHERT, S. RINDERLE-MA, P. DADAM: Flexibility in Process-Aware Information Systems. In: *Transactions on Petri Nets and Other Models of Concurrency* 2 (2009), S. 115–135

[RvS04] C. RICHTER-VONHAGEN, W. STUCKY: *Business-Process- und Workflow-Management*. B. G. Teubner Verlag, 2004

[Sch00] E. SCHICKER: *Datenbanken und SQL*. B. G. Teubner Verlag, 2000

[Sha05] Y. SHAFRANOVICH: *Common Format and MIME Type for Comma-Separated Value*. 2005. – Request for Comments: 4180

[SMR+08] H. SCHONENBERG, R. MANS, N. RUSSELL, N. MULYAR, W.M.P. VAN DER AALST: Process Flexibility: A Survey of Contemporary Approaches. In: J. DIETZ (Hrsg.), A. ALBANI (Hrsg.), J. BARJIS (Hrsg.): *CIAO! / EOMAS*, Springer, 2008 (LNBIP 10), S. 16–30

[SOS05] S. SADIQ, M. ORLOWSKA, W. SADIQ: Specification and validation of process constraints for flexible workflows. In: *Information Systems* 30 (2005), Nr. 5, S. 349–378

[TIB05] TIBCO SOFTWARE GMBH: *TIBCO Staffware Process Suite*. 2005. – Whitepaper

[Ull09] C. ULLENBOOM: *Java ist auch eine Insel*. Galileo Computing, 2009

[UML09] OBJECT MANAGEMENT GROUP: Unified Modeling Language Version 2.2. Object Management Group (OMG), Februar 2009. – Technischer Bericht

[W3C07] WORLD WIDE WEB CONSORTIUM: Web Services Description Language Version 2.0. 2007. – Technischer Bericht

[W3C08] WORLD WIDE WEB CONSORTIUM: Extensible Markup Language 1.0 (Fifth Edition). 2008. – Technischer Bericht

[WfM08] WORKFLOW-MANAGEMENT COALITION: XML Process Definition Language Version 2.1a. 2008. – Technischer Bericht

[WRRM08] B. WEBER, M. REICHERT, S. RINDERLE-MA: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. In: *Data and Knowledge Engineering* 66 (2008), September, Nr. 3, S. 438–466

[WSR09] B. WEBER, S. SADIQ, M. REICHERT: Beyond Rigidity - Dynamic Process Lifecycle Support: A Survey on Dynamic Changes in Process-aware Information Systems. In: *Computer Science - Research and Development* 23 (2009), Nr. 2

[WWRD07] B. WEBER, W. WILD, M. REICHERT, P. DADAM: ProCycle - Integrierte Unterstützung des Prozesslebenszyklus. In: *Künstliche Intelligenz* 20 (2007), Nr. 4, S. 1–15

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Christoph Mauroner Matrikelnummer: 518437

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Christoph Mauroner