Aspekte der Abbildung von Geschäftsprozessen, spezifiziert im Business Process Definition Metamodel, in BPEL4WS

Diplomarbeit

vorgelegt von

Kirsten Stöhr
(Matr-Nr. 8568060)

Leipzig, den 1. Februar 2005

Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik
Augustusplatz 10-11
04109 Leipzig

Gutachter:
Prof. Dr.-Ing. W.G. Spruth, Universität Leipzig
Michael Friess, IBM Entwicklung GmbH, Böblingen

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Die hier vorliegende Arbeit ist im Umfeld der Geschäftsprozessmodellierung einzuordnen und beschäftigt sich mit Aspekten der Abbildung von Geschäftsprozessen, spezifiziert im Business Process Definition Metamodel (BPDM), in die Business Process Execution Language for Web Services (BPEL4WS), speziell mit den bei der Abbildung auftretenden Komplikationen und Limitationen sowie mit damit verbundenen möglichen Problemlösungsansätzen. Ausgangspunkt für diese Arbeit bildet die Notwendigkeit der Spezifikation eines Ansatzes, der den zwischen Prozess-Designern und IT-Entwicklern heute existierenden „Graben“ [26] bezüglich der Entwicklung und Implementierung Geschäftsprozess-basierter Anwendungssysteme überbrückt.

1.1 Motivation

Heutzutage werden Geschäftsprozess-basierte Anwendungsysteme und die ihnen zugrundeliegenden Geschäftsprozessmodelle zunehmend komplexer, wodurch sich deren Implementierung zu einer immer größeren Herausforderung entwickelt [26]. Da auf Seiten der Geschäftsprozessmodellierung zahlreiche Methoden, Tools und Sprachen zur Beschreibung der zu implementierenden Geschäftsprozesse existieren, jedoch auf Seiten der Implementierung davon sich unterscheidende Methoden, Sprachen und Tools zur Charakterisierung der jeweiligen IT Artifakte verwendet werden, hat sich zwischen diesen im Kontext der Entwicklung Geschäftsprozess-basierter Anwendungssysteme ein signifikanter „Graben“ zwischen Designern von Geschäftsprozessen (im folgenden als Prozess-Designer bezeichnet), die maßgeblich für die Modellierung, Dokumentation und das Reengineering von Geschäftsprozessen zuständig sind und IT-Entwicklern, die diese implementieren und sich sehr stark mit deren technischen Aspekten auseinandersetzen, herausgebildet, den es, mittels geeigneten Methoden, gilt zu überwinden.

Hierzu bietet sich speziell der Modell-zentrierte Entwicklungs- und Implementations-Ansatz von IT-Systemen der Model Driven Architecture an, mit dem Geschäftsprozesse unabhängig einer konkreten Implementierungstechnologie anhand eines Plattform-unabhängigen Modells durch den Prozess-Designer spezifiziert werden, die über eines oder mehrere, näher an einer konkreten Ausführungsplattform liegende, Modelle in Plattform-abhängige Modelle, die den Geschäftsprozess bezüglich einer konkreten Implementierungsplattform spezifizieren unter zusätzlicher Angabe Plattform charakterisierender Informationen mittels Plattform Modellen, abgebildet werden, aus denen die notwendigen IT Artifakte erzeugt werden können.

Zur Spezifikation Plattform unabhängiger Modelle kann hierbei das Business Process Definition Metamodel als Sprache zur Beschreibung von Geschäftsprozessen verwendet werden, da sie zum einen unabhängig jeglicher Implementationsplattform ist, über eine große Modellmächtigkeit verfügt, von der

Object Management Group als einheitliche Sprache zur Geschäftsprozessmodellierung standardisiert wird, sowie sich auf Grund der Spezifikation als UML Profile nahtlos in den Model Driven Architecture Ansatz einfügt. Für den Bereich der Spezifikation Plattform abhängiger Modelle kann die Business Process Execution Language for Web Service bzw. eine dafür geeignetes Metamodell, beispielsweise das UML Profile für BPEL4WS, als Modellierunssprache Verwendung finden, da BPEL4WS mittlerweile eine sehr große Bedeutung im Umfeld ausführbarer Geschäftsprozesse erlangt hat.

Zur Umsetzung eines solchen Ansatzes unter Verwendung von BPDM und BPEL4WS besteht die Notwendigkeit, die Abbildung von BPDM Prozessen in BPEL4WS genau zu charakterisieren und etwaige mögliche Komplikationen und Limitationen bei der Abbildung als solches zu charakterisieren und für diese, wenn möglich, gegebenfalls Lösungsansätze zu spezifizieren.

1.2 Zielsetzung der Arbeit

Das Ziel der hier vorliegenden Arbeit ist es, die Abbildung von BPDM Prozessen in BPEL4WS im Kontext des Model Driven Architecture Ansatzes zum einen allgemein zu charakterisieren und dabei gezielt auftretende Komplikationen und Limitationen zu identifizieren sowie darauf aufbauend für die aufgezeigten Problemfelder unter Berücksichtigung der aktuellen BPEL4WS Spezifikation Version 1.1 und den bisher für BPEL4WS gemachten Erweiterungsvorschlägen [12] mögliche Lösungsansätze und Grenzen der Abbildung von BPDM in BPEL4WS aufzuzeigen.

Da sich zum Zeitpunkt der Entstehung dieser Arbeit das Business Process Definition Metamodel noch im Prozess der Standardisierung befand, beziehen sich alle hier gemachten Aussagen bzgl. BPDM auf [16] sowie der aktualisierten Submission von August 2004 [21].

1.3 Aufbau der Arbeit

Die hier vorliegende Diplomarbeit gliedert sich neben der Einleitung in 7 Kapitel, die im Folgenden kurz charakterisiert werden:

Kapitel 2 gibt eine kurze Einführung in die Thematik der Geschäftsprozessmodellierung, bei der einleitend auf den Begriff Geschäftsprozess eingegangen wird sowie anschließend einige typische Sprachen zur Geschäftsprozessmodellierung kurz vorgestellt werden. Kapitel 3 charakterisiert die Business Process Execution Language for Web Service als Sprache zur Beschreibung ausführbarer Geschäftsprozesse anhand der ausführlichen Beschreibung der einzelnen Sprachkonstrukte, sowie die mit BPEL4WS verbundene Web Service Architektur. Im Anschluss daran erfolgt in Kapitel 4 eine kurze Charakterisierung der Model Driven Architecture, sowie eine, für die nachfolgenden Kapitel notwendige, detaillierte Beschreibung der Sprachelemente des Business Process Definition Metamodels. Aufbauend auf Kapitel 3 und 4 wird in Kapitel 5 die Abbildung von BPDM in BPEL4WS charakterisiert sowie die bei der Abbildung auftretenden Limitationen und Komplikationen genauer beschrieben, für die in Kapitel 6 und 7 mögliche Lösungsansätze aufgezeigt werden. Hierbei beschreibt Kapitel 7 einen Lösungsansatz gesondert und vertiefend. Die Kapitel 5, 6 und 7 stellen das „Herzstück“ der Diplomarbeit dar. Kapitel 8 bildet den Abschluss der Arbeit, indem die wesentlichen Ergebnisse der Arbeit zusammengefasst werden.

Kapitel 2

Geschäftsprozesse und Sprachen zur Prozessbeschreibung

Da sich die hier vorliegende Arbeit mit Aspekten der Abbildung von in unterschiedlichen Sprachen spezifizierten Geschäftsprozessen beschäftigt, ist es notwendig Geschäftsprozesse und Sprachen zur Prozessbeschreibung einleitend zu charakterisieren. Hierzu werden im ersten Abschnitt Geschäftsprozesse und deren Beziehung zur Informationstechnologie sowie im darauffolgenden Abschnitt einige typische Sprachen zur Geschäftsprozessmodellierung näher dargestellt.

2.1 Geschäftsprozesse

Zur Zeit finden sich in der Literatur, aufgrund der Verwendung in verschiedenen Fachgebieten, unterschiedliche Definitionen des Begriffs Geschäftsprozess. Grundsätzlich kann aber zwischen zwei unterschiedlichen Interpretationen unterschieden werden. Zum einen die Interpretation aus Sicht des Business Process Reengineerings, die Geschäftsprozesse als Kernprozesse [1] ansieht, die das Leistungsprogramm eines Unternehmens darstellen und in ihrer Konsequenz einen Wert für den Kunden erbringen. Zum anderen im Kontext eines allgemeinen Prozessverständnisses bei dem Geschäftzsprozesse als betriebliche Prozesse verstanden werden, die einen wesentlichen Teil der Unternehmensleistung darstellen, beispielsweise Prozesse aus der Produktentwicklung und Marktforschung [44].

Eine davon abstrahierte Definition findet sich hierzu in [38]:

> *Ein Geschäftsprozess (Business Process) ist eine Folge von geschäftlichen Aktivitäten, die ein bestimmtes Ergebnis anstrebt. Geschäftsprozesse in ihrer Gesamtheit setzen die Geschäftsaufgabe um. DV-Systeme sollen die Durchführung der Geschäftsprozesse unterstützen und vereinfachen.*

D.h. ein Geschäftsprozess besteht aus einer Menge von Aktivitäten (Arbeitsschritte, Tasks), die in einer bestimmten Reihenfolge von Verarbeitungseinheiten abgearbeitet werden mit dem Ziel der Erbringung einer unternehmerischen Leistung. Hierbei zählen unter Verarbeitungseinheiten beispielsweise Menschen, Maschinen und Softwaresysteme.

Zusätzlich zu der im oberen Abschnitt dargestellten Definition werden Geschäftsprozesse durch die folgenden Merkmale charakterisiert [33]:

- sie haben einen Beginn und ein Ende.

[1] Prozesse, die für das Unternehmen essentiell sind.

- sie durchlaufen mehrere Organisationseinheiten.
- sie produzieren Leistungen oder Produkte.
- sie verfügen über klar definierte Eingabe- und Ausgabewerte.
- und sie sind anhand von Prozesszielen (z.B. Kosten, Zeit, Qualität) „meßbar“.

Klassisches Beispiel eines Geschäftsprozesses ist beispielsweise die Bearbeitung einer Schadensmeldung in einem Versicherungskonzern oder die Abwicklung eines Bestellvorgangs bei einem Versandhandel.

2.1.1 Geschäftsprozesse und Informationstechnologie

Durch den ständig wachsenden Wettbewerbsdruck werden Unternehmen immer mehr gezwungen die Effizienz ihrer Geschäftsprozesse zu steigern und diese an sich verändernde Marktsituationen anzupassen. Hierbei spielt die Informationstechnologie eine immer größer Rolle, indem beispielsweise viele der von Geschäftsprozessen durchgeführten Aktivitäten mit Hilfe von Anwendungssystemen unterstützt als auch IT-Systeme zur automatisierten Abarbeitung und Steuerung von Geschäftsprozessen eingesetzt werden.

Im Umfeld der Unterstützung von Geschäftsprozessen mittels Informationstechnologien können zwei wesentliche Arten von Geschäftsprozessen unterschieden werden: automatisierte und ausführbare Geschäftsprozesse.

Automatisierte Geschäftsprozesse (automated business processes), auch als sog. Workflows bezeichnet, charaktersisieren Geschäftsprozesse, die manuelle, von Personen auszuführende, als auch automatisierte, von DV-Systemen durchzuführende, Aktivitäten enthalten, die anhand eines Softwaresystems (Workflow Management System) in ihrer Ausführung gesteuert werden. Hierzu sind die zu unterstützenden Geschäftsprozesse anhand eines Modells spezifiziert indem die durchzuführenden Aktivitäten, deren benötigten Informationen sowie die Aktivitäten durchführenden Verarbeitungseinheiten anhand von Organisations- und Rollenmodellen beschrieben sind. Die Steuerung der Geschäftsprozesse gestaltet sich, indem das Workflow Management System anhand der Modelle die jeweils durchzuführenden Arbeitsschritte bestimmt, die an die jeweils dafür in Frage kommenden Verarbeitungseinheiten, von denen eine diese ausführt, weitergeleitet werden.

Demgegenüber existieren ausführbare Geschäftsprozesse (executable business processes), die komplett durch Softwaresysteme realisiert, ohne Interaktion mit Personen, ausführbar sind. Diese können beispielsweise als monolithische Anwendungen realisiert werden oder anhand von Software-Komponenten, die mittels Sprachen zur Prozessbeschreibung zu einem ausführbaren Geschäftsprozess gekoppelt werden, wodurch sich speziell Ablauf- und Business-Logik voneinander trennen lassen. Die somit realisierten Anwendungen werden als Geschäftsprozess-basierte Anwendungen (business process based applications) bezeichnet [24].

2.2 Sprachen zur Prozessbeschreibung

Damit Geschäftsprozesse zum Beispiel optimiert, analysiert, simuliert, dokumentiert, entworfen oder mit Hilfe von IT-Systemen in Form automatisierter Geschäftsprozesse unterstützt werden können, müssen diese in geeigneter Weise mittels Geschäftsprozessmodellen, unter Verwendung von Sprachen zur Prozessbeschreibung, die je nach Anwendungsgebiet (Simulation, Analyse, Dokumentation usw.) über unterschiedliche Fähigkeiten zur Beschreibung von Organisationsstrukturen, Ablaufstrukturen, Ressourcen, Informationen, Funktionen und Beziehungen von Geschäftsprozessen verfügen, spezifiziert werden.

Aufgrund der vielfältigen Einsatzgebiete der Geschäftsprozessmodellierung existiert mittlerweile eine Vielzahl von Sprachen, die sich zusätzlich zu den im vorherigen Absatz erwähnten Fähigkeiten, beispielsweise in der Notation, entweder graphisch oder textuell, in der Art der Beschreibung, beispielsweise fluss- oder zustandsorientiert, in der Möglichkeit zur Trennung von Kontroll- und Datenfluss oder in der Mächtigkeit zur Modellierung des Zeitverhaltens unterscheiden.

Um einen Überblick über Prozessbeschreibungs-Sprachen zu geben, werden im Folgenden drei Sprachen grob charakterisiert, die sich zum einen in ihrer Ausrichtung als auch beispielsweise in der Art der zugrundeliegenden Notation unterscheiden. Im ersten Abschnitt werden hierzu Petri-Netze, im Zweiten die Unified Modelling Language sowie abschließend eine XML basierte Sprache, die XML Process Definition Language, vorgestellt.

2.2.1 Petri-Netze

Eine Möglichkeit Geschäftsprozesse zu modellieren bieten Petri-Netze. Diese wurden im Jahre 1962 in der Dissertation von C.A. Petri zur Beschreibung und Analyse nebenläufiger Prozesse vorgeschlagen, die gegenüber anderen Modellierungssprachen anhand ihrer mathematischen Fundierung zur formalen Analyse (Erreichbarkeitsanalyse, Deadlockanalyse, Lebendigkeit) und Simulation von Geschäftsprozessen verwendet werden können. Aufgrund der Anwendung in unterschiedlichen Einsatzgebieten wurden Petri-Netze im Laufe der Zeit ständig erweitert, so dass zahlreiche Petri-Netz Typen entstanden, die sich mehr oder weniger gut zur Geschäftsprozessmodellierung eignen.

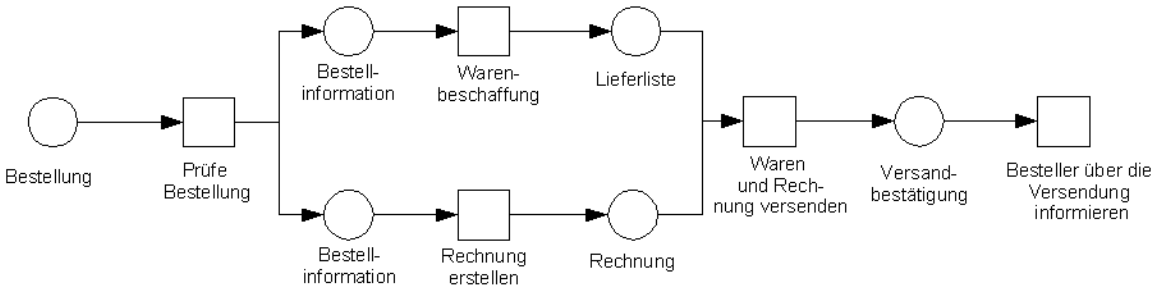

Abbildung 2.1: *Bestellprozess modelliert als Petri-Netz.*

In Abbildung 2.1 ist ein Modell eines einfachen Bestellprozesses als Petri-Netz dargestellt. Zu Beginn wird eine empfangene *Bestellung* auf ihre Gültigkeit geprüft. Konnte die Gültigkeit nachgewiesen werden, werden anschließend die in der *Bestellung* spezifizierten Waren beschafft und gleichzeitig eine *Rechnung* erstellt. Nach Abschluss der *Warenbeschaffung* und der *Rechnungserstellung* werden die Waren mit der *Rechnung* an den Kunden versendet, der zusätzlich über den erfolgreichen Versand anhand einer *Versandbestätigung* benachrichtigt wird.

Anhand dieses Beispiels sind die wesentlichen Elemente von Petri-Netzen erkennbar: sie bestehen aus einer Menge von Stellen, auch Bedingungen, Kanäle oder s-Elemente genannt, die als runde Kreise dargestellt werden und im Kontext von Geschäftprozessen mit „Datenspeichern“ vergleichbar sind, desweiteren aus einer Menge von Transitionen, symbolisiert in Form von Quadraten, vergleichbar mit Aktivitäten, die Verarbeitungsschritte charakterisieren. Hinzu kommen gerichtete Kanten, durch Pfeile verkörpert, die lediglich Transitionen und Stellen verbinden und den Kontroll- und Datenfluss bzw. die kausalen Abhängigkeiten zwischen Transitionen beschreiben, zusätzlich Marken (Token) auf Stel-

len, die Systemzustände bzw. im Kontext der Geschäftsprozessmodellierung Objekte repräsentieren, die von Kanten transportiert werden.

Zusätzlich zu den beschriebenen Elementen ist das Ausführungsmodell von Petri-Netzen wie folgt spezifiziert: Jede Transition besitzt eine Menge von, mit der Transition verbundenen, Stellen, die als Vor- und Nachbereich definiert sind. Ist der Vorbereich einer Transition mit Token belegt, so schaltet die Transition, vergleichbar mit der Aktivierung einer Aktivität, und emittiert in alle Stellen des Nachbereichs ein Token[2]. Aufgrund dieses Ausführungsmodells ist es für Petri-Netze möglich, die folgenden, in Abbildung 2.2 dargestellten, Kontrollflussstrukturen zu spezifizieren.

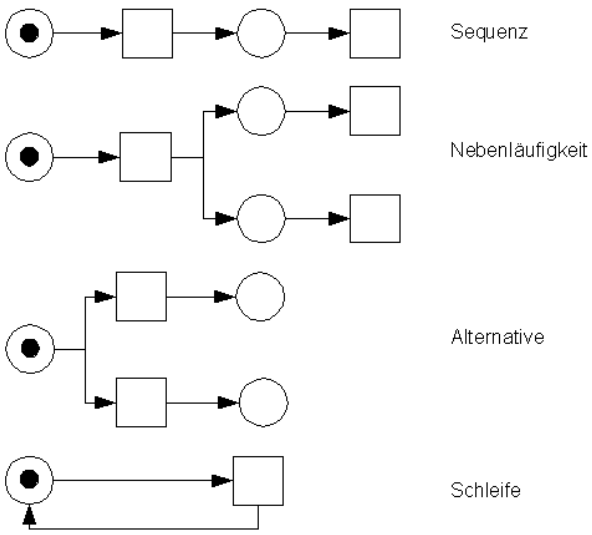

Abbildung 2.2: *Realisierung klassischer Kontrollflussstrukturen in Petri-Netzen.*

Der einfachste und gleichzeitig ausdrucksschwächste und damit für die Geschäftsprozessmodellierung nicht relevante Petri-Netz Typ sind die sog. Bedingungs /Ereignis-Netze (B/E-Netze). Diese bestehen aus den im vorherigen Absatz erwähnten Elementen, wobei Transitionen Ereignissen und Stellen Bedingungen enstsprechen. Bedingungen können jeweils nur genau ein Token aufnehmen, wodurch das Schalten einer Transition auf Grundlage der sicheren Transitionsregel, bei der der Nachbereich nicht belegt sein darf, geschieht. Nach erfolgreichem Schalten werden alle Bedingungen des Nachbereich der Transition mit Tokens belegt.

Abbildung 2.3 zeigt eine Erweiterung der B/E-Netze, die sogenannten Stellen-Transition-Netze (S/T-Netze). Sie ermöglichen gegenüber B/E-Netzen mehrere Tokens pro Stelle sowie individuelle Kantengewichte, die spezifizieren, wieviele Tokens aus den einzelnen Stellen zum schalten der Transitionen konsumiert werden müssen. Zusätzlich verfügen sie über die Möglichkeit Stellenkapazitäten anzugeben.

Eine weitere Entwicklung stellen die sog. Prädikat-Transitions-Netze (Pr/T-Netze) dar, die S/T-Netze um die Möglichkeit der Unterscheidung einzelner Tokens bzgl. ihrer Stellen sowie die Definition von Transitionsprädikaten erweitern. Dadurch sind sie der erste für die Geschäftsprozessmodellierung relvante Netztyp. Trotz dieser Tatsache existieren grundsätzliche Beschränkungen für die Modellierung

[2]In Abhängigkeit verschiedener Netztypen variiert die Anzahl der Tokens in Stellen die zur Aktivierung notwendig sind.

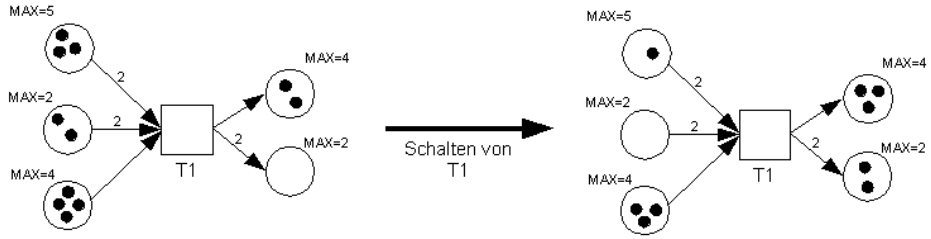

Abbildung 2.3: *Einfaches Stellen-Transitions-Netz vor und nach der Aktivierung.*

von Geschäftsprozessen: Es existiert kein explizites Typkonzept für Tokens (lediglich indirekte Typisierung möglich), keine spezifizierbaren Zugriffstrategien für Stellen, keine temporale Unterstützung sowie keine Möglichkeit zur Modellierung von Organisationseinheiten als auch zur Trennung zwischen Kontroll- und Datenfluss. In Abbildung 2.4 ist ein einfaches Pr/T-Netz dargestellt.

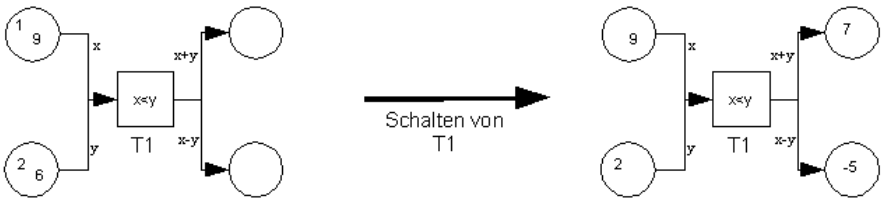

Abbildung 2.4: *Einfaches Prädikat-Transitions-Netz mit indirekter Typisierung und Transitionsprädikaten vor und nach der Aktivierung*

Daher wurden Pr/T-Netze zu sogenannten Funsoft-Netzen erweitert, die für die Geschäftsprozessmodellierung herangezogen werden können. Sie ermöglichen die Tokentypisierung, die Definition von Zugriffsstrategien auf Stellen, die Spezifizierung unterschiedlichen Schaltverhaltens von Transitionen sowie die Möglichkeit der Definition von Zeitverbrauch, Ausführungsmodus (manuell, automatisiert), Anzahl simultaner Ausführungen, die Verfeinerungen von Aktivitäten sowie die Trennung zwischen Daten- und Kontrollfluss. Der in Abbildung 2.1 dargestellte Bestell-Geschäftsprozess stellt eine vereinfachte Darstellung eines solchen Funsoft-Netzes dar.

Eine weitere Möglichkeit Geschäftsprozesse zu beschreiben besteht anhand der Unified Modelling Language.

2.2.2 Unified Modelling Language

Die Unified Modelling Language (UML) [35] [34], aktuell in der Version 2.0, ist eine von der Object Management Group (OMG) standisierte Sprache zur objekt-orientierten Spezifikation, Visualisierung und Konstruktion von Modellen für Softwaresysteme. Sie spezifiziert eine Menge verschiedener Modellelemente, die in unterschiedlichen Diagrammtypen kombiniert zur grafischen Beschreibung unterschiedlicher Aspekte eines System eingesetzt werden. Beispielsweise zur Beschreibung der statischen Struktur von IT-Systemen anhand von Klassendiagrammen oder deren dynamisches Verhalten mittels Aktivitätsdiagrammen.

Die von UML zur Verfügung gestellten Sprachelemente sind anhand eines Metamodells [3] beschrieben, dass mit einer Teilmenge an Elementen von UML, die durch das Meta Object Facility (MOF) Meta-Metamodell spezifiziert sind, selber ausgedrückt ist und die abstrakte Syntax der jeweiligen Modellelemente spezifiziert. Man kann sich das in ungefähr so wie die deutsche Sprache vorstellen, die sich selbst erklärt: Der Duden regelt welche Wörter in der Sprache enthalten sind und was sie bedeuten und wie sie zu Sätzen zusammengesetzt werden [11]. So wird beispielsweise im UML-Metamodell spezifiziert, dass Klassen (*Class*), Operationen (*Operation*) und Attribute (*Property*) als Sprachelemente existieren und Klassen über mehrere Operationen und Attribute verfügen können. Die folgende Abbildung zeigt hierzu einen Ausschnitt aus dem UML-Metamodell.

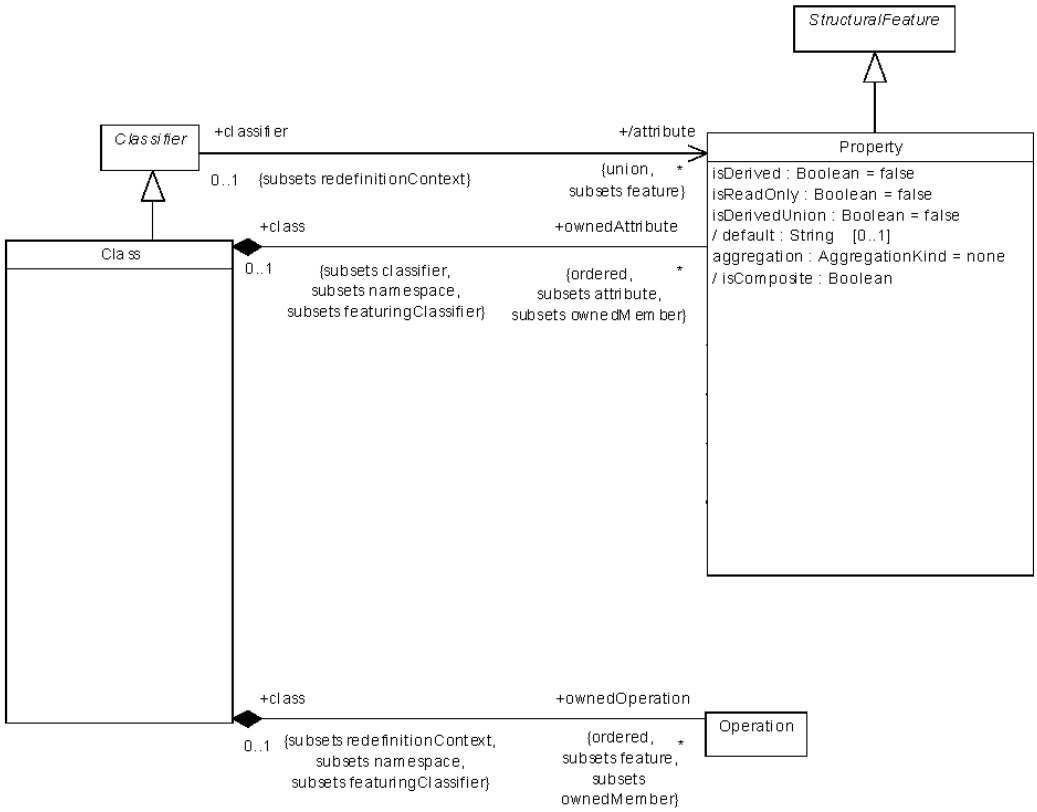

Abbildung 2.5: *Abstrakte Syntax der Definition von Klassen im UML-Metamodell.*

Da das UML-Metamodell lediglich die reine Syntax der jeweiligen Sprachelemente spezifiziert wird deren Semantik zusätzlich anhand der natürlichen Sprache beschrieben. Darüber hinaus sind den einzelnen Metamodellelementen graphische Notationen zugeordnet, anhand deren sie zur graphischen Modellierung verwendbar sind.

UML eignet sich nicht nur zur Modellierung von IT-Systemen sondern kann aufgrund seiner Mächtigkeit auch zur Spezifikation von Geschäftsprozessen eingesetzt werden. Hierbei können besonders Aktivitätsdiagramme eingesetzt werden, anhand derer der Prozessfluss und das Verhalten von Geschäftsprozessen in vergleichbarer Weise zu Petri-Netzen spezifiziert werden kann. Darüber hinaus können mit Hilfe des UML Profile-Mechanismus zusätzliche Stereotypen spezifiziert werden, anhand

[3] Ein Metamodell ist ein Modell eines Modells. Jedes Modell ist eine Instanz des Metamodells.

derer beispielsweise Ressourcen- und Organisationsaspekte mittels UML-Klassendiagrammen abgebildet werden können.

An dieser Stelle sei auf Kapitel 4 verwiesen, in dem die Geschäftsprozessmodellierung mittels UML anhand des Business Process Definition Metamodels vertieft wird.

2.2.3 XML basierte Sprachen

Wie bereits der Name andeutet handelt es sich hierbei um Sprachen, mittels derer auf Basis von XML Geschäftsprozesse beschrieben werden. Dies hat den Vorteil, dass die Prozessdefinitionen leichter verarbeitet und manipuliert als auch auch zwischen verschiedenen Modellierungstools ausgetauscht werden können. Im Folgenden soll die von der Workflow Management Coalition (WfMC) verabschiedete XML Process Definition Language (XPDL) [48], als Beispiel einer XML basierten Sprache, näher dargestellt werden.

XML Process Definition Language

XPDL stellt im wesentlichen ein Grundgerüst an Modellierungselementen für die Prozessbeschreibung, speziell von automatisierten Geschäftsprozessen, zur Verfügung. Anhand Abbildung 2.6, welches das Prozess-Metamodell zeigt, sind die wesentlichen Elemente erkennbar, die mittels äquivalenten XML Schema Typen XPDL definieren und zur Modellierung verwendet werden können.

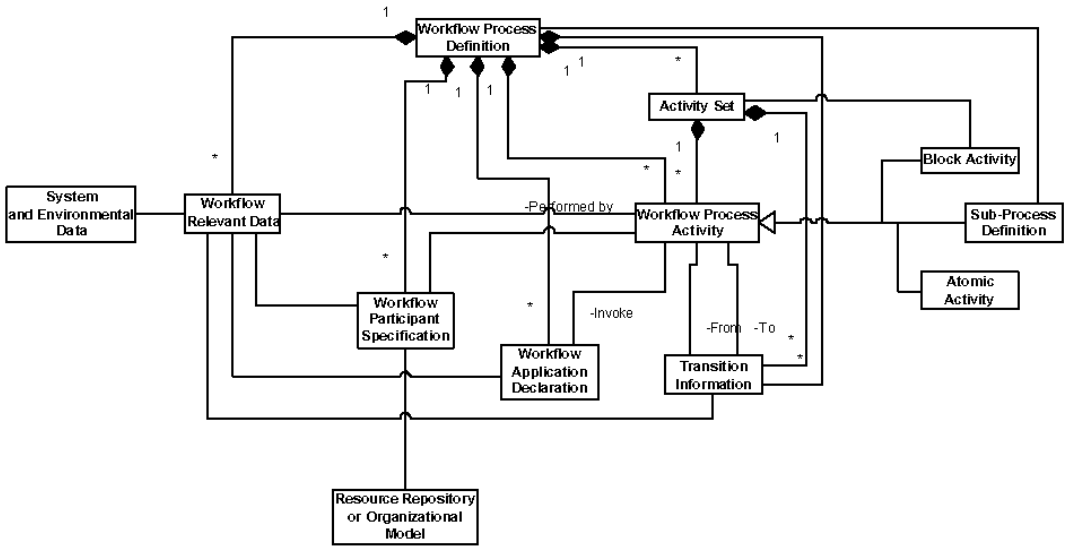

Abbildung 2.6: *Das Workflow Prozess Definition Metamodel für XPDL.*

Anhand des Metamodells wird ein Geschäftsprozess als Instanz des Workflow Process Definition (`WorkflowProcesses`) Elements beschrieben, welches aus einer Menge von Workflow Process Activity- (`Activity`), Transition Information- (`Transition`), Workflow Application Declaration- (`Application`), Workflow Relevant Data- (`DataField`) sowie Activity Set (`ActivitySets`) Elementen besteht.

Das nachfolgende Listing zeigt die Struktur eines mittels XPDL modellierten Geschäftsprozesses in Äquivalenz zum Prozess-Metamodell.

```
<WorkflowProcess Name="" ID="">
    <ProcessHeader>
    </ProcessHeader>
    <RedefineableHeader/>*
    <FormalParameters/>*
    <Participants/>*
    <DataFields>*
        <DataField>
        </DataField>
    </DataFields>
    <Applications/>*
    <ActivitySet/>*
    <Activities>*
        <Activity Id="">*
            {<Route/> | <Implementation/> | <BlockActivity>}
            <TransitionRestrictions>*
                <TransitionRestriction>
                   q21 {<Join/> | <Split/>}
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
    </Activities>
    <Transitions>*
        <Transition Id="" From="Activity ID" To="Activity ID">?
            <Condition/>*
        </Transition>
    </Transitions>
</WorkflowProcess>
```

Im wesentlichen beschreiben `Activities` die Arbeitsschritte (`Activity`), die innerhalb eines Geschäftsprozesses durchgeführt werden und die mittels `Transition` Elementen untereinander verbunden sind. Transitionen können zusätzliche Bedingungen sog. Transition Conditions besitzen, die darüber entscheiden, ob nach Aktivierung der Quellaktivität die Zielaktivität der Transition auszuführen ist. Existieren mehrere eingehende oder ausgehende Transitionen so besteht die Möglichkeit über `TransitionRestiction` Elemente, z.B. zusätzliche Kontrollfluss Beschränkungen in Form von `Join` Elementen für eingehende Transitionen mit AND- oder XOR Semantik oder `Split` Elementen für ausgehende Transitionen mit ebenfalls AND- oder XOR Semantik zu definieren.

Die Datenmodellierung erfolgt durch `DataField` Elemente, die Prozessvariablen entsprechen und die zur Definition von Transitionsbedingungen und als Übergabeparameter Anwendung finden. `Applications` werden benutzt, um die vom Geschäftsprozess aufgerufenen Anwendungen und Tools in Form von `Application` Elementen zu spezifizieren. Darüberhinaus besteht die Möglichkeit mittels `ActivitySet` Elementen mehrere Aktivitäten und zugehörige Transitionen, unter der Bedingung das keine Transition aus dem `ActivitySet` herausführen, zu gruppieren.

Neben Petri-Netzen, UML und XPDL existieren weitere Sprachen zur Prozessbeschreibung, die jeweils auf Grund ihrer Ausrichtung beispielsweise zur Spezifikation automatisierter oder ausführbarer Geschäftsprozesse eingesetzt werden. Speziell im Umfeld ausführbarer Geschäftsprozesse hat sich mittlerweile die Business Process Execution Language for Web Service, etabliert die automatisierte Geschäftsprozesse mittels der Verknüpfung von Web Services spezifiziert und zunehmend an Bedeutung gewinnt.

Kapitel 3

Business Process Execution Language for Web Services

Heutzutage ist es mittels Web Services möglich, mit beliebigen Anwendungssystemen, die Web Services zur Verfügung stellen, anhand standardisierter, hersteller- und plattform-unabhängiger Protokolle zu interagieren ohne deren interne Struktur zu kennen, womit sie sich speziell zur Unterstützung von Geschäftsprozessen, speziell zur Realisierung ausführbarer Geschäftsprozessen, durch deren Komposition (Web Service Orchestration) besonders eignen. Daher wurden in diesem Kontext zu Beginn verstärkt propritäre, auf XML basierte Sprachen zur Komposition von Web Services, wie die Web Service Flow Language (WSFL) [30] von IBM oder XLANG [39] von Microsoft spezifiziert, deren Konzepte in einer standardisierten Sprache zur Beschreibung ausführbarer Geschäftsprozesse, der Business Process Execution Language for Web Services (BPEL4WS), die in diesem Kapitel näher charakterisiert wird, subsummiert wurden.

Da BPEL4WS aufgrund der starken Beziehung zu Web Services viele der in diesem Bereich spezifizierten Standards verwendet, wird im ersten Abschnitt einleitend die Web Service Architektur kurz charakterisiert, bevor im darauffolgenden Abschnitt die wesentlichen Merkmale und Sprachelemente von BPEL4WS beschrieben werden.

3.1 Web Service Architektur

Die Web Service Architektur stellt eine Realisierung des Konzepts der Service Oriented Architecture dar, bei der mit den von Web Services zur Verfügung gestellten Schnittstellen, die in einer standardisierten Sprache, in der Regel WSDL, beschrieben werden, unter Verwendung von Web-Standards wie HTTP und XML, lose gekoppelte und miteinander interagieren Anwendungssysteme realisierbar werden. Im Folgenden werden hierzu die wesentlichen Elemente der Web Service Architektur charakterisiert, angefangen bei einer kurzen Darstellung der Service Oriented Architecture, über die allgemeine Beschreibung von Web Services bis hin zu den im Web Service Umfeld verwendeten Standards SOAP, WSDL und UDDI.

3.1.1 Service Oriented Architecture

Das Ziel jeglicher Softwarentwicklung besteht in der Realisierung zuverlässiger Software, die leicht zu warten als auch wiederverwendbar, d.h. im Kontext anderer Software, nutzbar ist. Um diese Ziele zu erreichen wurde das Konzept der Service Oriented Architecture (SOA) entwickelt.

Die Service Oriented Architecture läßt sich am einfachsten anhand der von ihr spezifizierten Rollen, die jeweils Komponenten, die diese Architektur implementieren, einnehmen, charakterisieren: Die zentrale Rolle spielt hierbei der Dienstanbieter (sog. Service-Provider), der Dienste (sog. Services) anhand von Schnittstellen zur Verfügung stellt, die er selbst implementiert und die von Dienstnutzern (sog. Service Requestern) in Anspruch genommen werden. Hierbei sind Dienstnutzer und Dienstanbieter untereinander lose gekoppelt, wodurch ebenfalls andere Dienstnutzer diesen Dienst in Anspruch nehmen können. Die von einem Dienstanbieter zur Verfügung gestellten Dienste können hierbei anhand einer in computerverarbeitbaren Form spezifizierten Schnittstellenbeschreibung von jedem Dienstnutzer in Erfahrung gebracht werden. Darüber hinaus existiert neben Dienstanbieter und -nutzer ein Dienstverzeichnis (sog. Service Directory), dass die von Dienstanbietern veröffentlichten Schnittstellen in katalogisierter Form verwaltet und anhand dessen Dienstnutzer, die spezielle Dienste in Anspruch nehmen wollen den jeweils den Dienst zur Verfügung stellenden Dienstanbieter, der beim Dienstverzeichnis registriert ist, ermitteln können.

Die Web Service Architektur kann als eine Realisierung der Service Oriented Architecture angesehen werden, bei der Web Services Service Provider darstellen, die eine Menge von Diensten, die in einer computerverarbeitbaren Form, in der Regel anhand der Web Service Description Language (WSDL), beschrieben werden, beliebigen Anwendungen, den Service Requestern zur Verfügung stellen, die diese beispielsweise unter Verwendung XML basierter Protokolle, z.B. dem Simple Object Acess Protocol (SOAP), in Anspruch nehmen. UDDI stellt hierbei das Service Directory dar, dass die von Web Services zur Verfügung gestellten Dienst verwaltet. Die folgende Abbildung verdeutlicht diesen Zusammenhang zwischen der SOA und der Web Service Architektur.

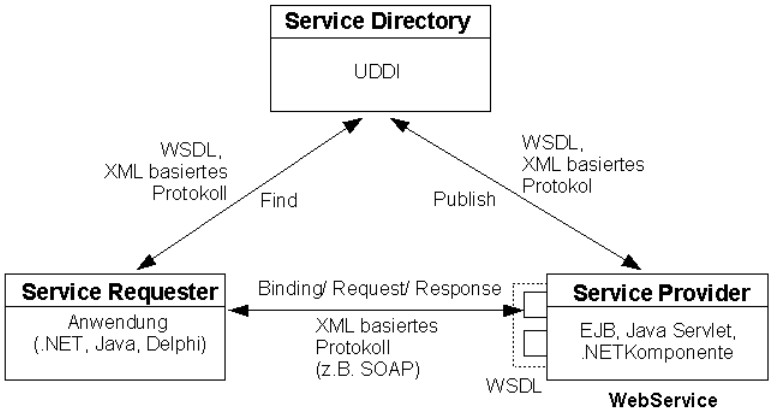

Abbildung 3.1: *Komponenten der WebService Architektur im Kontext der SOA.*

Um die Web Service Architektur genauer zu charakterisieren werden im Folgenden deren Komponenten näher dargestellt.

3.1.2 Web Service

In der Literatur finden sich zahlreiche Definitionen des Begriffs Web Service, die sich zum Teil sehr stark unterscheiden und jeweils verschiedene Aspekte der zugrundeliegenden Technologien mehr oder weniger stark betonen. Deshalb wird an dieser Stelle eine davon abstrahierte, auf die wesentlichen Eigenschaften ausgerichtete Definition vorgestellt, die Web Services wie folgt charakterisiert [20]:

> *A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language (WSDL)).*

D.h. bei Web Services handelt es sich um Softwaresysteme, die mittels verschiedenster Softwaretechnologien beispielsweise Java Servlets oder Enterprise Java Beans (EJB) realisiert sein können, die Schnittstellen zur Verfügung stellen, die anhand eines von Maschinen verarbeitbaren Formats, in der Regel WSDL, spezifiziert sind und auf die mittels einer von ihnen unterstützten Möglichkeit zur Interaktion über ein Netzwerk zugegriffen werden kann.

Diese Definition läßt offen, welche Protokolle zur Interaktion bzw. welche Sprachen zur Beschreibung der Schnittstellen von Web Services verwendet werden. Prinzipiell sind alle Sprachen, die die in der Definition genannten Anforderungen erfüllen, dafür geeignet. Mittlerweile haben sich jedoch in diesem Umfeld Standards, wie WSDL oder SOAP, etabliert, die zur Beschreibung und Interaktion mit Web Services genutzt werden und die im Folgenden kurz erläutert werden.

3.1.3 Web Service Description Language

Die Web Service Description Language (WSDL) [14] ist eine auf XML basierte Sprache zur Beschreibung von Web Services, die in der Version 1.1 in diesem Abschnitt kurz charakterisiert wird.

Die WSDL beschreibt einen Web Service als eine Menge von Schnittstellen, die mehrere mögliche Interaktionen anhand von Operationen mit einer Anwendung spezifizieren, auf die über definierte Adressen unter Verwendung festgelegter Kommunikationsprotokolle zugegriffen werden kann. Die Spezifikation erfolgt hierbei in zwei voneinander unabhängigen Teilen: Im ersten abstrakten Teil werden die Schnittstellen, deren Operationen sowie deren Ein- und Ausgabenachrichten unabhängig einer konkreten Adresse und eines Kommunikationsprotokoll, damit unabhängig einer konkreten Implementierung, spezifiziert, die im zweiten Teil anhand der Festlegung der Bindung an ein Kommunikationsprotokoll, das beispielsweise die Codierung von Ein- und Ausgabenachrichten von Operationen festlegt, und der Spezifikation der Adressen bezüglich einer Implementierung konkretisiert werden.

Hierfür definiert WSDL folgende Elemente: `definitions`, `types`, `messages`, `portType`, `binding`, `port` und `service`, die im weiteren kurz charakterisiert werden.

Das `definitions`-Element ist das Wurzel-Element eines jeden WSDL-Dokuments, dass die im Folgenden beschriebenen Elemente enthält.

Mittels `types`-Elementen werden die Datentypen spezifiziert, die bei der Interaktion verwendet werden. Hierbei ist WSDL nicht auf ein festes Typ-System beschränkt. In der Regel kommt als Typ-System XML Schema zum Einsatz.

Die `message`-Elemente beschreiben die Nachrichten bzw. Daten die von Web Services empfangen bzw. versendet werden ohne Bezug zu deren Implementierung. Sie bestehen aus benannten logischen Abschnitten `parts`, die anhand von `type`-Elementen typisiert sind.

`portType`-Elemente spezifizieren die von Web Services unterstützten Schnittstellen unabhängig deren Implementierung. Schnittstellen fassen eine Menge von Operationen (`operation`), die sich auf eine spezifizierte `input`- und `output-message`s beziehen, zusammen. Hierbei werden anhand von Operationen vier verschiedene Interaktionsmuster zwischen Web Service und Client unterstützt: klassisches Request/Response, bei der der Client eine Nachricht sendet und der Web Service mit einer korrelierten Nachricht anwortet, One-Way, bei der der Client lediglich eine Nachricht an den Service sendet, Notification bei der der Web Service eine Nachricht an den Client sendet und Solicited-Request, bei der der Web Service dem Client eine Nachricht sendet, der diese mit einer Nachricht beantwortet.

`binding`-Elemente legen für die von `portType`-Elementen beschriebenen Operationen und deren Nachrichten die für den Zugriff zu verwendenden Protokolle und Datenformate fest. WSDL definiert Standard-Bindings für SOAP, HTTP und MIME. Zusätzlich existieren weitere beispielsweise für Java Message Service (JMS) [41].

`port`-Elemente legen für ein `binding` die Adresse fest, über die anhand der im `binding` spezifizierten Protkolle und Formate, auf die dem `binding` zugeordneten `portTyp` Operationen, zugegriffen werden kann.

`service`-Elemente beschreiben die von einem Web Service angebotenen Dienst anhand der Zusammenfassung einer Menge von `port`s.

Um die Zusammenhänge zwischen den wesentlichen Elementen von WSDL zu verdeutlichen, zeigt das nachfolgende Listing die Spezifikation eines einfachen Bestell-Services (`BestellService`) anhand von WSDL.

```
<definitions name="BestellService"
          ....
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="BestellTypRequest">
              <complexType>
                  <sequence>
                      <element name="firma"     type="string"/>
                      <element name="produktID" type="int"/>
                  </sequence>
              </complexType>
           </element>
           <element name="BestellTypResponse">
              <complexType>
                  <sequence>
                      <element name="Rechnung" type="string"/>
                      <element name="status"   type="string"/>
                  </sequence>
              </complexType>
           </element>
       </schema>
    </types>

    <message name="BestellRequestMsg">
        <part name="body" element="BestellTypRequest"/>
    </message>

    <message name="BestellResponseMsg">
        <part name="body" element="BestellTypResponse"/>
    </message>

    <portType name="BestellServicePortType">
        <operation name="bestellProdukt">
            <input message="BestellRequestMsg"/>
            <output message="BestellResponseMsg"/>
        </operation>
    </portType>
```

```
    <binding name="BestellServiceSoapBinding" type="BestellServicePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="bestellProdukt">
            <soap:operation soapAction="http://example.com/BestellService"/>
            <input>
                <soap:body use="literal"/>
           </input>
           <output>
                <soap:body use="literal"/>
           </output>
        </operation>
    </binding>

    <service name="BestellService">
        <port name="BestellServicePort" binding="tns:BestellServiceSoapBinding">
            <soap:address location="http://example.com/BestellService"/>
        </port>
    </service>
</definitions>
```

Wie im Absatz über das `binding`-Element dargelegt, können verschiedene Protokolle für die Kommunikation zwischen Client und Web Service verwendet werden. Speziell hierfür hat sich in den letzten Jahren das Simple Object Access Protocol als quasi Standard etabliert, welches im obigen Beispiel als Kommunikationsprotokoll spezifiziert wurde und im folgenden Abschnitt kurz charakterisiert wird.

3.1.4 Simple Object Access Protocol

Das Simple Object Access Protocol (SOAP) [49], momentan in der Version 1.2, ist ein, auf XML basiertes, Kommunikationsprotokoll, welches für den Austausch von strukturierten, getypten Informationen, in Form von Nachrichten, zwischen Knoten innerhalb einer dezentralen und verteilten Umgebung spezifiziert wurde und speziell im Web Service Bereich Anwendung findet [49].

Es handelt sich um ein zustandsloses Protokoll, welches zum Austausch von One-way Nachrichten zwischen SOAP Sender und SOAP Empfänger konzipiert ist, mit dem sich darüber hinaus komplexere Interaktionsformen , beispielsweise Request/Response oder Request/Multiple Response anhand der Kombination von einfachen Nachrichten und zugrundeliegenden Transportprotokollen oder durch zusätzliche anwendungsspezifische Informationen innerhalb von Nachrichten realisieren lassen. SOAP macht prinzipiell keine Aussagen über die Semantik der zu transportierenden Daten, das Routing von Nachrichten, Verlässlichkeit des Transfers oder der Überwindung von Firewalls. Es stellt lediglich ein Framework zum Nachrichtenaustausch aufbauend auf vorhandenen Transportprotokollen wie HTTP oder SMTP dar [42].

Abbildung 3.2: *Aufbau von SOAP Nachrichten.*

SOAP Nachrichten sind relativ einfach aufgebaut. Sie werden anhand des `Envelope`, dem Wurzelelement, eingeleitet und bestehen aus zwei Teilen: einem `Header` und einem `Body`, dargestellt in Abbildung 3.2. Der `Header` ist optional und wird zum Transport von Kontrollinformationen, beispielsweise Verarbeitungshinweisen, genutzt. Der `Body` ist demgegenüber Pflicht, da er die eigentlichen, von der Nachricht transportierten Informationen enthält, bei denen es sich beispielsweise um Daten-Objekte handeln kann, die anhand der SOAP Encoding Regel serialisiert wurden oder um entfernte Operationsausfrufe (RPC) zum Beispiel von Web Services.

Obwohl SOAP grundsätzlich ein unabhängiges Kommunikationsprotokoll ist, wird es in der Regel mittels Transportprotokollen wie HTTP, SMTP oder FTP übertragen. Speziell HTTP und SMTP sind als Standard-Bindings für SOAP spezifiziert. Besonders im Web Service Umfeld erfreut sich SOAP über HTTP zunehmender Beliebtheit. Hierbei werden SOAP Request Nachrichten via HTTP Requests und SOAP Response Nachrichten mittels HTTP Response übertragen. Dabei kommen für HTTP Requests sowohl die GET- als auch POST-Methode zum Einsatz, wobei von den meisten Servern die POST-Methode, aufgrund der beschränkten Zeichenanzahl der GET-Methode, bevorzugt wird.

Das nachfolgende Listing zeigt einen solches HTTP Request, der eine SOAP Nachricht zum Aufruf der `OrderProducts` Operation des `SimpleOrder` Web Service enthält. Hieran sind die wesentlichen Bestandteile von SOAP-Nachrichten, wie `Envelope`- und `Body`-Element, erkennbar.

```
POST /BestellService HTTP/1.1
Host: www.example.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 379

<SOAP:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Header>
  </SOAP:Header>
  <SOAP:Body>
      <m:bestellProdukt xmlns:m="Some-URI">
          <firma>DIS</firma>
          <produktID>36435</produktID>
      </m:bestellProdukt>
  </SOAP:Body>
</SOAP:Envelope>
```

Neben WSDL und SOAP findet im Web Service Umfeld ein weiterer Standard Anwendung, UDDI, auf den im Folgenden kurz eingegangen wird.

3.1.5 Universal Description Discovery and Integration

Um die von Web Services angebotenen Dienste nutzen zu können, müssen die jeweiligen Dienstbeschreibungen in Form von WSDL Dokumenten bzw. deren Lokalisierung bekannt sein. Ist dies der Fall, stellt die Nutzung kein Problem dar. Was aber, wenn WSDL Dokumente bzw. deren Lokalisierung nicht bekannt sind, sondern lediglich Eigenschaften wie beispielsweise der Name einer Operation? In diesem Fall würde keine Möglichkeit bestehen den jeweiligen Dienst ausfindig zu machen. Daher ist ein zusätzlicher Verzeichnisdienst, vergleichbar den „Gelben Seiten“, notwendig, anhand dessen Web Services, die spezifische Dienste anbieten, ermittelt werden können.

Die Universal Description, Discovery and Integration (UDDI) [4] Spezifikation beschreibt einen solchen Dienst basierend auf einem verteilten Datenbestand, in dem Informationen über Unternehmen

und deren Dienste in einer Datenbasis, der UDDI Business Registration, an verschiedenen UDDI Knoten abgelegt werden. Diese Informationen bestehen aus drei Komponenten: „white pages“ beinhalten Anschrift, Kontaktinformationen sowie bekannte Identifizierungen; „yellow pages“ die industrielle Kategorisierung basierend auf einer Standard-Taxonomie und „green Pages“ die technischen Details der Services, die das Unternehmen zur Verfügung stellt speziell eine Referenz auf die zugehörigen WSDL-Dokumente.

Der Zugriff auf den UDDI Server erfolgt anhand standardisierten Web Services Schnittstellen, die Funktionen für das Auffinden als auch zum Veröffentlichen von Services umfassen.

3.2 Business Process Execution Language for Web Services

Die Business Process Execution Language for Web Services (BPEL4WS) [13], momentan in der Version 1.1, ist eine auf XML basierte Sprache zur Beschreibung ausführbarer Geschäftsprozesse, deren Verhalten durch die Interaktion mit den von Partnern zur Verfügung gestellten Web Services, die mittels WSDL spezifiziert sind, realisiert wird. Hierfür spezifiziert BPEL4WS eine Menge von Elementen, mit denen beispielsweise die mit einem Prozess interagierenden Partner, die Reihenfolge der Aufrufe von durch Partner zur Verfügung gestellten Web Service Operationen sowie die zwischen den Aufrufen zu transportierenden Daten spezifiziert werden.

BPEL4WS Prozesse interagieren nicht nur mit einer Menge von Partnern über Web Services Schnittstellen, sondern sie verkörpern nach außen ebenfalls Web Services, die durch diese implementiert werden und mit denen über anhand von WSDL spezifizierten Schnittstellen interagiert werden kann (Abbildung 3.3). Alle Interaktionen werden hierbei, um die Einfachheit und Wiederverwendbarkeit von BPEL4WS Prozessen zu gewährleisten, „abstrakt“ durch den Bezug auf PortTyp- anstatt auf Port-Definitionen konkreter Web Services beschrieben, wodurch speziell Binding- und Deployment-Aspekte außerhalb der Spezifikation von Prozessmodellen belassen werden [29].

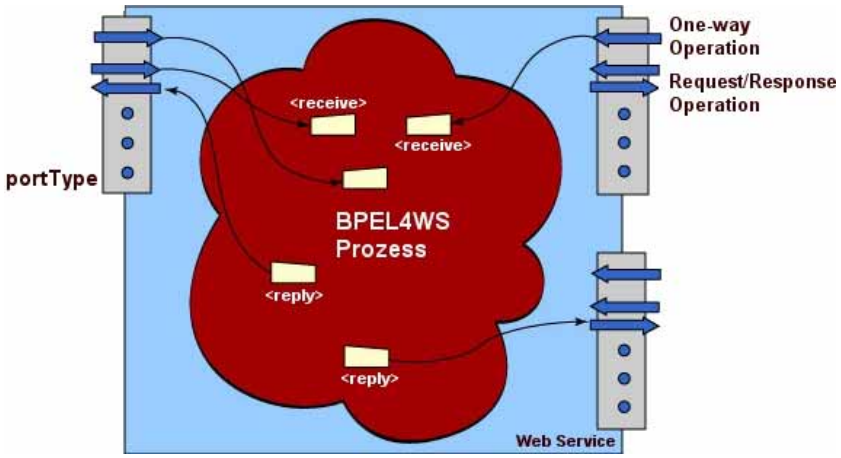

Abbildung 3.3: *Blick auf einen Web Services, der anhand eines BPEL4WS Prozesses implementiert wird.*

Neben der Beschreibung ausführbarer Geschäftsprozesse mittels executable BPEL4WS Prozessen, die innerhalb einer BPEL4WS Ausführungsumgebung ausführbar sind, bietet BPEL4WS ebenfalls die Möglichkeit Geschäftsprotokolle (business protocols) zwischen Partnern anhand nicht ausführbarer

sog. abstrakter Prozesse zu spezifizieren, die den nach außen sichtbaren Ablauf von zwischen Partnern existierenden Kollaborationen mittels den zwischen ihnen auszutauschenden Nachrichten charakterisieren. Jeder abstrakte Prozess beschreibt hierbei das Verhalten eines in einem Geschäftsprotokoll involvierten Partners aus dessen lokaler Sicht anhand der Spezifikation des Ablaufs von Aktionen zum Versenden als auch Empfangen von Nachrichten. Somit charakterisieren abstrakte Prozesse das nach außen sichtbare Verhalten eines Partners (sog. external view), wohingegen ausführbare BPEL4WS Prozesse deren internes Verhalten (sog. internal view) spezifizieren. Zur Beschreibung abstrakter als auch ausführbarer Prozesse kommen in BPEL4WS jeweils die gleichen Konzepte zur Anwendung.

Durch die starke Beziehung zu Web Services setzt BPEL4WS auf zahlreiche XML Spezifikationen auf: WSDL 1.1, XML Schema 1.0 und XPath 1.0. Hierbei werden anhand von WSDL Nachrichten und XML Schemas die von BPEL4WS Prozessen verwendeten Datenmodelle charakterisiert. XPath dient als Sprache zur Beschreibung von Datenmanipulationen sowie WSDL zur Spezifikation aller externen Ressourcen und Partner als Web Services.

Um BPEL4WS genauer zu charakterisieren, wird im Folgenden der in Abbildung 2.1 dargestellte Bestell-Geschäftsprozess als ausführbarer BPEL4WS Prozess einleitend dargestellt. Im Anschluss daran wird die Struktur von BPEL4WS Prozessen sowie die damit verbunden Sprachkonstrukte näher spezifiziert.

3.2.1 Beispiel

Der in Abbildung 2.1 dargestellte Bestell-Prozess interagiert aus Sicht von BPEL4WS mit vier Partnern: dem Kunden, der die Bestellung auslöst, der Warenbeschaffung, die die Warenliste generiert, der Rechnungsabteilung, die die Rechnung erstellt und der Versandabteilung, die anhand der Warenliste die bestellten Waren samt der Rechnung versendet und eine Versandbestätigung an den Kunden zurückschrickt. Die Interaktion erfolgt hierbei über die von den Partnern zur Verfügung gestellten Web Service Schnittstellen. Abbildung 3.4 zeigt in Form eines UML Klassendiagramms die Partner des BPEL4WS Bestell-Prozesses sowie die bei Interaktionen ausgetauschten Daten.

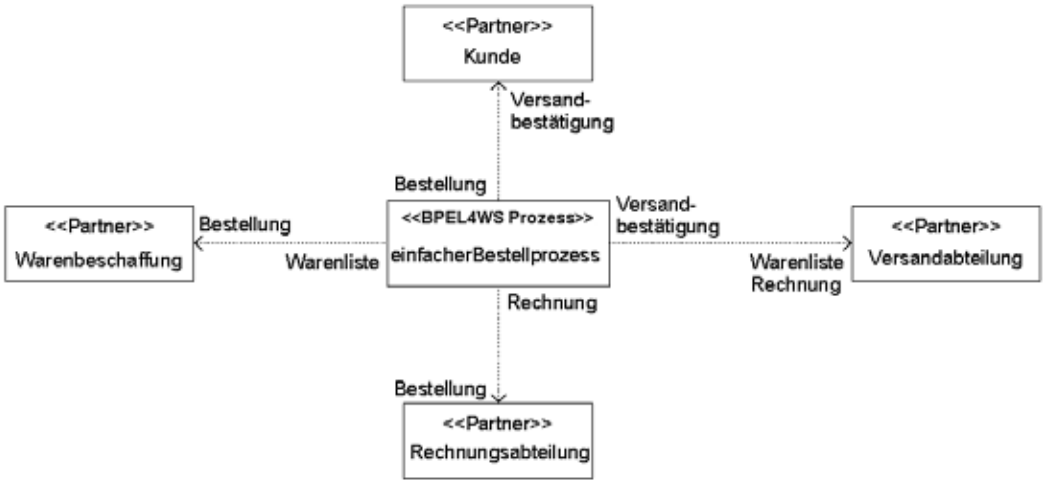

Abbildung 3.4: *Darstellung der Partner des SimpleOrder BPEL4WS Prozesses in Form eines UML Klassendiagramms.*

Die von den Partnern, mit Ausnahme des Kunden, zur Verfügung gestellten Web Services Schnitt-

stellen sind anhand von WSDL beschrieben. Das nachfolgende Listing zeigt jeweils Ausschnitte der WSDL des Warenbeschaffungs-, Rechnungsabteilungs- und Versandabteilungs-Service sowie des durch den BPEL4WS Prozess implementierten Web Service. Darin sind jeweils die auszutauschenden Nachrichten als auch die angebotenen Schnittstellen-Operationen anhand von `message`- und `portType`-Deklarationen erkennbar.

```
<message name="Warenliste">
    <part name="warenliste" type="..."/>
</message>
<portType name="WarenbeschaffungPT">
    <operation name="ermittleWarenliste">
        <input  message="Bestellung"/>
        <output message="Warenliste"/>
    </operation>
</portType>

<message name="Rechnung">
    <part name="rechnung" type="..."/>
</message>
<portType name="RechnungsabteilungPT">
    <operation name="erstelleRechnung">
        <input  message="Bestellung"/>
        <output message="Rechnung"/>
    </operation>
</portType>

<message name="VersandwarenRechnung"
    <part name="warenliste" type="..."/>
    <part name="rechnung"   type="..."/>
</message>
<message name="Versandbestaettigung">
    <part name="versandbestaettigung" type="..."/>
</message>
<portType name="VersandabteilungPT">
    <operation name="versendeWaren">
        <input  message="VersandwarenRechnung"/>
        <output message="Versandbestaettigung""/>
    </operation>
</portType>

<message name="Bestellung"
    <part name="BestellungID" type="xsd:integer"/>
    <part name="Bestellungen" type="..."/>
</message>
<portType name="BestellprozessPT">
    <operation name="bestelleWaren">
        <input  message="Bestellung"/>
        <output message="Versandbestaettigung"/>
    </operation>
</portType>
```

Der eigentliche Bestell-Prozess mit seinen einzelnen Arbeitsschritten ist als BPEL4WS Prozess im nachfolgenden Listing dargestellt. Dieser beschreibt zu Beginn anhand von `partnerLink`s die mit dem Prozess interagierenden Partner sowie anhand von `variables`, die vom Prozess zur Datenhaltung verwendeten Variablen. Anschließend werden die vom Prozess durchzuführenden Arbeitsschritte anhand von Aktivitäten spezifziert. Zu Beginn wird vom Prozess die `sequence`-Aktivität ausgeführt,

die als Struktur-Aktivität die sequentielle Abarbeitung der in ihr enthaltenen Aktivitäten spezifiziert. Daher wird nachfolgend mittels der `receive`-Aktivität auf den Eingang einer Bestellung des Kunden durch Aufruf der `bestelleWaren` Operation des BPEL4WS Prozesses gewartet. Sobald ein Kunde die `bestelleWaren` Operation aufruft, wird die `receive`-Aktivität beendet und die die Bestellung aufnehmende `BestellungV` Variable als Eingabevariable (`inputVariable`) für die durch die `flow`-Aktivität anschließend parallel auszuführenden `invoke`-Aktivitäten, die die `ermittleWarenliste`-Operation des Warenbeschaffungs- und `erstelleRechnung`-Operation des Rechnungsabteilungs-Partners zur Bestimmung der Warenliste und zur Erstellung der Rechnung aufruft, verwendet. Die von den Web Service Aufrufen zurückgelieferte Warenliste und Rechnung werden in Variablen gehalten, deren Inhalt anschließend anhand der `assign`-Aktivität der Variable `WarenRechnung` zugewiesen wird, die als Eingabevariable für die darauffolgende `invoke`-Aktivität, die die `versendeWaren`-Operation des Versandabteilungs Partners zum Versenden der durch die Variable spezifizierten Waren und der Rechnung aufruft, verwendet wird und eine Versandbestätigung zurückliefert, die mittels `reply`-Aktivität dem Kunden als Rückgabewert des Aufrufs der `bestelleWaren`-Operation zurückgegeben wird.

```
<process name="einfacherBestellprozess" isAbstract="false">
    <partnerLinks>
        <partnerLink name="ProzessKunde" .../>
        <partnerLink name="ProzessWarenbeschaffung" .../>
        <partnerLink name="ProzessRechnungsabteilung" .../>
        <partnerLink name="ProzessVersandabteilung" .../>
    </partnerLinks>

    <variables>
        <variable name="BestellungV"   messageType="Bestellung"/>
        <variable name="WarenlisteV"   messageType="Warenliste"/>
        <variable name="RechnungV"     messageType="Rechnung"/>
        <variable name="WarenRechnung" messageType="VersandwarenRechnung"/>
        <variable name="Versandbest"   messageType="Versandbestaettigung"/>
    </variables>

    <sequence>
        <receive partnerLink="ProzessKunde" portType="BestellprozessPT"
                 operation="bestelleWaren" variable="BestellungV"/>
        <flow>
           <invoke partnerLink="ProzessWarenbeschaffung" portType="WarenbeschaffungPT"
                   operation="ermittleWarenliste"
                   inputVariable="BestellungV" outputVariable="WarenlisteV"/>
            <invoke partnerLink="ProzessRechnungsabteilung" portType="RechnungabteilungPT"
                    operation="erstelleRechnung"
                    inputVariable="Bestellung" outputVariable="RechnungV"/>
        </flow>
        <assign>
            <copy><from  variable="WarenListeV"   part="warenListe"/>
                  <to    variable="WarenRechnung" part="warenListe"/></copy>
            <copy><from  variable="RechnungV"     part="rechnung"/>
                  <to    variable="WarenRechnung" part="rechnung"/></copy>
        </assign>
        <invoke partnerLink="ProzessVersandabteilung" portType="VersandabteilungPT"
                operation="versendeWaren"
                inputVariable="WarenRechnung" outputVariable="VersandBest"/>
        <reply partnerLink="ProzessKunde" portType="BestellprozesspT
               operation="bestelleWaren" variable="VersandBest"/>
    </sequence>
</process>
```

Zusätzlich sind die zu verwendeten Datentypen beispielsweise in Form von XML Schema Definitionen zu spezifizieren, was an dieser Stelle vernachlässigt wurde.

3.2.2 Struktur von BPEL4WS Prozessen

Anhand des obrigen Beispiels ist bereits die Grundstruktur von BPEL4WS Prozessen bzw. BPEL4WS Dokumenten zu erahnen, die anhand des nachfolgenden Listing genau charakterisiert ist:

```
<process name="ncname" isAbstract="true|false">
    <partnerLinks>        ... </partnerLinks>
    <partners>            ... </partners>
    <variables>           ... </variables>
    <correlationSets>     ... </correlationSets>
    <faultHandlers>       ... </faultHandlers>
    <compensationHandler> ... </compensationHandler>
    <eventHandlers>       ... </eventHandlers>

    activity
</process>
```

Sie werden mit dem `process`-Element, welches anhand des `name`-Attributs den zu modellierenden Geschäftsprozess benennt, dessen Typ (ausführbar oder abstrakt) mittels des `isAbstract`-Attributs bestimmt wird, eingeleitet, dessen untergeordnete Elemente bis auf eine Aktivität (`activity`) optional sind, die entweder durch Basis-Aktivitäten, die elementare Operationen, wie beispielsweise den Aufruf einer Web Service Operation eines Partners charakterisieren oder Struktur-Aktivitäten, die den Kontrollfluss von Basis-Aktivitäten spezifizieren, bestimmt ist.

3.2.3 BPEL4WS Spracherweiterung

Die im Kontext von BPEL4WS verwendbaren Elemente sind dabei in ihrer Struktur keineswegs festgelegt, sondern können durch die Fähigkeit der Spracherweitung von BPEL4WS mit zusätzlichen Attributen oder untergeordneten Elementen anderer XML Namensräume erweitert und in ihrer Semantik präzesiert werden. Beispiel einer solchen Erweiterung könnte die Einführung eines `subProzess`-Attributes des `process`-Elements sein, welches zusätzlich die Unterscheidung zwischen regulären ausführbaren Prozessen oder ausführbaren Subprozessen erlaubt, dessen Verwendung syntaktisch folgendermassen aussieht:

```
<process xmlns:ws="..." ... ws:subProcess="true" ...>
```

An dieser Stelle sei bemerkt, dass alle in einem Prozess verwendeten Erweiterungen von der jeweiligen BPEL4WS Ausführungsumgebung für deren Ausführung unterstützt werden müssen.

3.2.4 Lebenszyklus von BPEL4WS Prozessen

Da BPEL4WS Prozesse langlebig und zustandsbehaftet und Prozessdefinitionen nur Modelle ausführbarer Geschäftsprozesse sind, werden sie zur Ausführungszeit instanziiert.

Die Erzeugung von Prozessinstanzen erfolgt implizit anhand von Aktivitäten (`receive` o. `pick`), die externe Nachrichten über die durch den Prozess definierten Web Service Schnittstellen empfangen und anhand derer von der BPEL4WS Ausführungsumgebung bestimmt, wird ob eine neue Instanz erzeugt werden soll oder nicht. Die zur Erzeugung von Prozessinstanzen zu verwendeten Aktivitäten werden hierbei anhand des `createInstance=true` Attributs charakterisiert.

Prozessinstanzen werden beendet, wenn die Aktivitäten, die das Verhalten von Prozessen bestimmen, abgearbeitet wurden (normale Beendigung), wenn ein Fehler den Prozess-Gültigkeitsbereich erreicht, der entweder behandelt oder nicht behandelt wird, oder wenn von Prozessinstanzen die `terminate`-Aktivität zur explizit Beendigung ausgeführt wurde.

3.2.5 Partner Charakterisierung

Geschäftsprozesse interagieren zu unterschiedlichen Zeitpunkten mit verschiedenen Partnern um Leistungen von diesen in Anspruch zu nehmen, wobei bezüglich der Interaktion jeder Partner eine spezielle Rolle gegenüber dem anderen inne hat. Typischer Weise handelt es sich bei den Interaktionen um Punkt-zu-Punkt Interaktionen. In BPEL4WS erfolgt die Interaktion mit Partnern über die von ihnen zur Verfügung gestellten Web Service Schnittstellen anhand von Operationsaufrufen, wobei durch WSDL lediglich die Funktionalität der von den Partnern zur Verfügung gestellten Web Services spezifiziert wird. Da WSDL ausschließlich die von Partnern gebotene Funktionalität nicht aber die Beziehungen zwischen Partnern und Prozessen charakterisiert, existiert in BPEL4WS das Konzept von Partner Links, mit denen die Prozess-Partner festgelegt werden und darüberhinaus zur Charakterisierung der Interaktionen zwischen BPEL4WS Prozessen und Partnern sowie deren dabei eingenommenen Rollen dienen. Die mit Partner Links spezifizierten Konversationen sind Punkt-zu-Punkt Konversationen zwischen Prozessen und Partnern, die die Form der Beziehung zwischen beiden mittels den an der Interaktion beteiligten Nachrichten und PortTypen in beide Richtungen sowie den dabei von den Partnern bei der Interaktion eingenommenen Rollen beschreiben. Die Definition von Partnern erfolgt in BPEL4WS hierbei in zwei Schritten: Der Partner Link Typ- und der eigentlichen Partner Link-Spezifikation.

Partner Link Typen

Ein Partner Link Typ charakterisiert eine Interaktion zwischen zwei Web Services (Partner), indem er die Rollen definiert, die jeder Web Services in der Interaktion spielt sowie die Porttypen charakterisiert, die jeweils von den Web Services zum Empfangen und Versenden von Nachrichten im Kontext der Interaktion zur Verfügung gestellt werden [13]. Sie spezifizieren lediglich die Interaktionstypen bzw. Interaktionsmuster sowie die damit verbundenen Anforderungen bzgl. der für die Interaktionen notwendigen Schnittstellen und die Schnittstellen inne habenden Rollen, nicht aber eine konkrete Interaktion zwischen zwei Web Services. Das nachfolgende Listing zeigt die allgemeine Struktur der Definition von Partner Link Typen:

```
<partnerLinkType name="name">
    <role name="name">
        <portType name="qname"/>
    </role>
    <role name="name">
        <portType name="qname"/>
    </role>
</partnerLinkType>
```

Das `name`-Attribut charakterisiert den Interaktionstyp, die Rollen, spezifiziert durch das `role`-Element, jeweils die an der Interaktion beteiligten Rollen sowie die von ihnen zur Interaktion zur Verfügung zustellenden Porttypen. Jede Rolle spezifiziert hierbei exakt einen Porttyp.

Da Web Service-Interaktionsmuster existieren bei denen keinerlei Anforderungen an einen der beiden Web Services bzgl. der Realisierung eines speziellen Porttyps vorhanden sind, wird in diesen Fällen lediglich eine Rolle spezifiziert, wodurch beliebige Web Services mit dem die Rolle inne habenden Web Service interagieren können, ohne das spezielle Anforderung von diesen zu erfüllen sind.

Partner Link Typen unterscheiden sich zu anderen BPEL4WS Konstrukten insofern, dass sie im Kontext von WSDL Dokumenten anhand des WSDL Extension Mechanismus spezifiziert werden und ein wesentliches Element zur Spezifikation der zwischen Web Services existierenden Interaktionen sind und somit als losgelöst von BPEL4WS betrachtet werden können.

Partner Links

Die mit BPEL4WS Geschäftsprozessen interagierenden Web Services werden in Form von Partner Links spezifiziert, welche anhand von Partner Link Typen charakterisiert werden [13]. Gleichzeitig beschreiben sie die Interaktionen die zwischen Prozessen und ihren Partnern ablaufen, wobei sie die dabei von Prozessen bzw. ihren Partnern im Kontext der Interaktion eingenommenen Rollen charakterisieren. Prinzipiell können mehrere Partner Links anhand des gleichen Partner Link Typs charakterisiert werden. Die Syntax von Partner Link Definitionen ist anhand des nachfolgenden Listings gezeigt:

```
<partnerLinks>
    <partnerLink name=""   partnerLinkType="qname"
                 myRole="" partnerRole=""/>
</partnerLinks>
```

Das `name`-Attribut benennt den zu spezifizierenden Partner Link, der `partnerLinkTyp` den auf sich die Interaktion beziehenenden Interaktionstyp, `myRole`, die Rolle des BPEL4WS Prozesses und `partnerRole`, die Rolle des Partners im Kontext der Interaktion bezogen auf den spezifizierten Partner Link Typ.

An dieser Stelle sei angemerkt, dass, bevor Operationen von Partnern via Partner Links aufgerufen werden können, zusätliche Binding- und Lokations-Informationen, über die, im Kontext der Partner Links, zu verwendenden Partner Services, benötigt werden, die im Bereich des Deployments von BPEL4WS Prozessen spezifiert werden und die, wie bereits in der Einleitung erwähnt, nicht Gegenstand der Spezifikation von BPEL4WS sind.

Da zwischen BPEL4WS Prozessen und deren Partnern nicht nur eine Interaktion abläuft sondern mehrere verschiedene in denen die gleichen Partner jeweils unterschiedliche Rollen spielen, können in BPEL4WS alle einen Partner betreffenden Partner Links anhand der `partner` Spezifikation unter einem Namen zusammengefasst werden. Das nachfolgende Listing zeigt die Syntax von Partner Definitionen:

```
<partners>
    <partner name="">
        <partnerLink name="">+
    </partner>
</partners>
```

Grundsätzlich sind `partner` Spezifikationen optional, die darüberhinaus nicht alle Partner Links bzgl. eines Partners zusammenfassen müssen.

3.2.6 Datenbehandlung

Da Geschäftsprozesse in der Regel langlebig und zeitintensiv sind, ist es, für die Interaktionen mit ihren Partnern, notwendig, Daten über einen längeren Zeitraum zur Verfügung zu halten. BPEL4WS bietet hierfür Variablen, die anhand der von BPEL4WS Prozessen empfangenen Nachrichten mit Daten belegt als auch untereinander zu neuen Variablen kombiniert und zur Steuerung des Verhaltens von Prozessen verwendet werden. Hierzu spezifiziert BPEL4WS eine Menge an Modellierungskonstrukten, mit denen Daten aus Variablen extrahiert und anderen Variablen zugewiesen als auch Ausdrücke über Variablen definiert werden können.

Variablen

Variablen dienen im Kontext von BPEL4WS dazu, Daten die von Partnern empfangen oder zu Partnern verschickt werden, sowie Statuswerte, die zu Steuerung des Kontrollflusses notwendig sind, zu halten und damit den Zustand von BPEL4WS Prozessen zu charakterisieren. Sie werden aufgrund der strengen Typsicherheit von BPEL4WS vor ihrer ersten Verwendung deklariert, wobei die Deklaration immer bezüglich eines bestimmten Gültigkeitsbereichs (Abschnitt 3.2.10) erfolgt. Hierbei kann aufgrund der Unterscheidung zwischen einem globalen (Prozess)- und lokalen Gültigkeitsbereichen zwischen globalen und lokalen Variablen unterschieden werden. Zur Typisierung von Variablen werden entweder WSDL Nachrichten, einfache XML Schema Typen, wie beispielsweise `xsd:int` oder `xsd:string`, oder XML Schema Elemente verwendet. Anhand des nachfolgenden Listing ist die Syntax von Variablen-Deklarationen erkennbar:

```
<variables>
    <variable name="name"   messageType="qname"?
              type="qname"? element="qname"?/>+
</variables>
```

Das `name`-Attribut benennt die zu deklarierende Variable, anhand dessen diese im Prozess verwendet wird. Die `messageType`-, `type`- und `element`-Attribute spezifizieren den Typ von Variablen, wobei genau eines dieser Attribute angegeben werden muss. Hierbei referenziert das `messageType`-Attribut auf eine WSDL Nachricht, das `type`-Attribut auf einen einfachen XML Schema Typ und das `element`-Attibut auf ein XML Schema Element.

Datenzuweisungen

Beim Umgang mit Variablen ist es notwendig, Daten von Variablen anderen Variablen zuzuweisen (sog. Assignment). Hierzu werden in BPEL4WS `assign`-Aktivitäten verwendet, mit denen Daten zwischen Variablen kopiert als auch Variablen mit neuen Daten, die anhand von Ausdrücken spezifiziert werden, belegt werden können. `assign`-Aktivitäten charakterisieren hierbei eine oder mehrere elementare Daten-Zuweisungen, die anhand von `copy`-Elementen beschrieben werden, von denen jedes einen typ-kompatiblen Wert einer Quelle (`from-spec`) an ein Ziel (`to-spec`) kopiert. Die Syntax zeigt das nachfolgende Listing:

```
<assign>
    <copy>+
         from-spec
         to-spec
    </copy>
</assign>
```

Hierbei können für `from-spec` und `to-spec` die folgenden Varianten verwendet werden:

```
from-spec:
    <from variable  ="name" part="name"? query="queryString"?/>
    <from expression="general-exp"/>
to-spec:
    <to variable="name" part="name"? query="queryString"?/>
```

Die erste Variante beschreibt die Möglichkeit direkt den Inhalt von Variablen gleichen Typs zuzuweisen, wobei die Zuweisung bis auf `part` und Element-Ebene, anhand der `query`-Spezifikation, von durch WSDL Nachrichten getypten Variablen verfeinert werden kann. Demgegenüber ermöglicht die zweite Variante einfache Berechnungen über Variablen durchzuführen wie beispielsweise die Erhöhung einer Zählvariable, dargestellt im nachfolgenden Listing. Die hierzu verwendete `getVariableData`-Funktion wird im nachfolgenden Abschnitt genauer charaktersiert.

```
<assign>
    <copy>
        <from expression="bpws:getVariableData('zaehler')+1"/>
        <to variable="zaehler"/>
    </copy>
</assign>
```

Neben den oben genannten Varianten werden in [13] weitere genannt, die an dieser Stelle nicht weiter charakterisiert werden sollen, da sie eher eine untergeordnete Rolle spielen.

Ausdrücke

In vielen Situationen ist es notwendig über Variablen beispielsweise logische Bedingungen, speziell im Kontext der Kontrollflusssteuerung, zu formulieren als auch Berechnungen durchführen zu können. Hierfür bietet BPEL4WS die im folgenden dargestellten Typen von Ausdrücken [13] an:

- Boolsche-Wert Ausdrücke, verwendet beispielsweise zur Definition von `while`- oder `switch`-Bedingungen,
- Deadline-Wert Ausdrücke, benutzt zur Spezifikation von Zeitpunkten, beispielsweise bei `onAlarm`- und `wait`-Aktivitäten,
- Duration-Wert Ausdrücke, eingesetzt zur Charakterisierung von Zeitintervallen, beispielsweise bei `onAlarm`- und `wait`-Aktivitäten,
- und allgemeine Ausdrücke, angewendet beispielsweise im Kontext von `assign`-Aktivitäten,

die in einer durch das `expressionLanguage`-Attributs des `<process>` Elements festgelegten Sprache, in der Regel XPath 1.0, in BPEL4WS Prozessen spezifiziert werden.

Im Falle der Verwendung von XPath Ausdrücken bietet BPEL4WS zusätzliche XPath-Funktionserweiterungen, mit denen aus XPath Ausdrücken auf prozessspezifische Informationen zugegriffen werden kann. Eine dieser Erweiterungen ist die `bpws:getVariableData('variableName', 'partName'?, 'locationPath'?)` Funktion, die den Zugriff auf Daten von BPEL4WS Variablen aus XPath Ausdrücken ermöglicht. Hierbei spezifiziert der `variableName` die zu referenzierende Variable, der optionale `partName` den zu verwendenden `part` der durch eine WSDL Nachricht getypten Variable und der optionale `locationPath` einen XPath-Pfadausdruck beginnend mit /, der ein bestimmtes Element des `part` spezifiziert, beispielsweise in einem durch ein komplexes XML Schema Element[1] typisierten `part` ein Element oder alle Elemente eines durch ein XML Schema Element spezifizierten Arrays.

3.2.7 Korrelationen

Aufgrund der Tatsache das BPEL4WS Prozesse zur Ausführungszeit instanziiert werden, besteht die Möglichkeit, dass gleichzeitig mehrere Instanzen ein und desselben Prozesses aktiv sind und damit Nachrichten, die zwischen Prozessen und Partnern ausgetauscht werden, neben der richtigen Lokation auch an die richtige Instanz weitergeleitet werden müssen. Um die richtige Prozessinstanz zu addressieren verwendet BPEL4WS einen Mechanismus, der anhand spezifizierter Nachrichtenteile eine Korrelation vornimmt, mithilfe derer Nachrichten spezifischen Instanzen zugeordnet werden. Hierzu werden sog. Korrelationsmengen definiert, die aus einer Menge von Message Properties bestehen, die Teile zu übertragender Nachricht identifizieren, die vergleichbar mit Variablen sind und die während der Ausführungszeit von Prozessinstanzen mit den durch sie charakterisierten Nachrichtenteilen einer Nachricht einmalig initialisiert werden und als Alias für die Prozessinstanz fungieren. Die Initialisierung erfolgt hierbei durch Aktivitäten, wie `receive`, `reply` oder `invoke`, anhand der von ihnen zu

[1] Ein XML Schema Element welches anhand eines komplexen XML Schema Typs typisiert ist.

sendenden oder empfangenden Nachrichten. Beim Empfang einer die Korrelationsmenge spezifizierten Nachrichtenteile enthaltenden Nachricht kann anhand eines Tests auf Gleichheit der Werte der Korrelationsmenge und der Nachricht, aufgrund der Zuordnung Korrelationsmenge-Prozessinstanz, festgestellt werden, an welche Prozessinstanz die empfangene Nachricht weitergeleitet werden soll. Grundsätzlich werden Korrelationsmengen sowohl in ausführbaren als auch abstrakten BPEL4WS Prozessen eingesetzt, um die Prozess-relevanten als auch im Kontext der Spezifikation von Geschäftsprotokollen notwendigen Adressierungen von Partner-Prozessen zu spezifizieren. Das nachfolgende Listing zeigt die Syntax zur Definition von Korrelationsmengen:

```
<correlationSets>
    <correlationSet name="name" properties="name-list"/>+
</correaltionSets>
```

Das `name`-Attribut benennt die Korrelationsmengen und das `properties`-Attribut spezifiziert die Menge zu verwendenter Message Properties. Für die Charakterisierung der Syntax der Spezifikation von Message Properties sei an dieser Stelle auf [13] verwiesen.

Die Verwendung von Korrelationsmengen soll anhand zweier miteinander interagierender BPEL4WS Prozesse, einem Verkäufer- und einem Käuferprozess, erläutert werden: Der Käuferprozess sendet zu Beginn eine Bestellung in Form einer Nachricht an den Verkäuferprozess und initialisiert gleichzeitig eine Korrelationsmenge, die die Bestell-ID der Bestellung als Message Property enthält. Der Verkäufsprozess empfängt diese Nachricht, erzeugt eine Rechnung und verpackt diese mit der Bestell-ID in einer Nachricht die er an den Käuferprozess zurückschickt. Aufgrund der initialisierten Korrelationsmenge kann auf Seiten des Käuferprozesses der, die zur Rechnung korrespondierende Bestellung, versendete Käuferprozess ermittelt werden, an den die Rechnung weitergeleitet wird. Ohne Korrelationsmengen hätte im Kontext mehrere miteinander interagierender Käufer- und Verkäuferprozessinstanzen nicht bestimmt werden können, welcher Käufer- mit welchem Verkäuferprozess interagiert. Das nachfolgende Listing zeigt die Umsetzung auf Seiten des Käuferprozesses in BPEL4WS:

```
<message name="BestellungMsg">
    <part name="BestellID" type="xsd:int"/>
    ...
</message>

<message name="RechnungMsg">
    <part name="BestellID" type="xsd:int"/>
    ...
</message>

<variables>
    <variable name="Bestellung" messageType="BestellungMsg"/>
    <variable name="Rechnung"   messageType="RechnungMsg"/>
</variables>

<correlationsSets>
    <correlationSet name="BestellungCS" properties="BestellungMsgIDProp"/>
</correaltionSets>
...
<invoke operation="sendeBestellung" inputVariable="Bestellung" ... >
    <correlations>
        <correlation set="BestellungCS" initiate="yes" pattern="out"/>
    </correlations>
</invoke>
...
<receive operation="empfangeRechnung" variable="Rechnung" ...>
    <correlations>
```

```
        <correlation set="BestellungCS"/>
    </correlations>
</receive>
...
```

Zu Beginn wird die Korrelationsmenge `BestellungCS` definiert, die sich lediglich anhand der `BestellungMsgIDProp` auf den `BestellID` Part der `BestellungMsg` Nachricht bezieht. Zu Beginn wird anhand der `invoke`-Aktivität die Bestellung an den Verkäuferprozess versendet und die `BestellungCS` Korrelationsmenge anhand des `correlation`-Elements durch Angabe des `initiate=yes`-Attributes initialisiert. Das `pattern`-Attribut spezifiziert hierbei, ob die Korrelationsmenge anhand der eingehenden und ausgehenden Nachricht initialisiert werden soll, was in diesem Fall überflüssig ist, da es sich um ein one-way `invoke` handelt, welches lediglich eine Nachricht versendet. Anschließend wird beim Empfang der Rechnung durch die `receive`-Aktivität mittels `correlations`-Element die empfangene Rechnung an der Korrelationsmenge `BestellungCS` korreliert, wodurch die Rechnung an genau die Prozessinstanz weitergeleitet wird deren Korrelationsmenge, damit die BestellID, mit der in der Rechnung ebenfalls spezifizierten BestellID übereinstimmt.

3.2.8 Basis-Aktivitäten

Basis Aktivitäten beschreiben die von BPEL4WS Prozessen auszuführenden Arbeitsschritte. Hierzu zählen `invoke`-Aktivitäten, für den Aufruf von Web Service Operationen von Partnern, `receive`- und `reply`-Aktivitäten , zum Reagieren auf den Aufruf Prozess spezifischer Service Operationen, `assign`-Aktivitäten, für die Verarbeitung von Variablen, die bereits im Abschnitt 3.2.6 charakterisiert wurden, sowie `throw`-, `wait`- und `empty`-Aktivitäten, die im Folgenden näher dargestellt werden.

invoke

`invoke`-Aktivitäten charakterisieren den Aufruf von durch Partner zur Verfügung gestellte Web Service Operationen über einen spezifizierten Partner Link. Die Ein- bzw. Ausgabedaten werden hierbei anhand von BPEL4WS Variablen, die als Ein- bzw. Ausgabevariablen fungieren und durch WSDL Nachrichten typisiert sind, der `invoke`-Aktivität übergeben bzw. von ihr zurückgeliefert. Neben dem Aufruf klassischer synchroner Request/Response Operationen können ebenfalls asynchrone One-way Operationen aufgerufen werden, wobei die Aufrufe sich lediglich syntaktisch anhand der Spezifikation einer Ausgabevariable unterscheiden. Da synchrone WSDL Operationsaufrufe Fehlernachrichten zurückliefern können, werden von BPEL4WS sog. Fault Handler spezifiziert, mit denen diese abgefangen werden können. Im Kontext der Fehlerbehandlung besteht zusätzlich die Möglichkeit sog. Kompensations-Aktivitäten zu spezifizieren. Das nachfolgende Listing zeigt die Struktur von `invoke`-Aktivitäten:

```
<invoke partnerLink="name" portType="qname" operation="name"
        inputVariable="name" outputVariable="name">
    <correlations>
        <correlation set="name" initiate="yes|no"
                     pattern="in|out|out-in"/>
    </correlations>
    <catch faultName="qname" faultVariable="name">
        activity
    </catch>
    <catchAll>
        activity
    </catchAll>
    <compensationHandler>?
        activity
    </compensationHandler>
</invoke>
```

Das `partnerLink`-Attribut charakterisiert hierbei den zu verwendenden Partner Link, das `portType`-Attribut, die zu verwendende Schnittstelle des Partners, das `operation`-Attribut, die aufzurufende Operation der Schnittstelle, `inputVariable` und `outputVariable`, die der Operation übergebenen Ein- bzw. von ihr zurückgelieferten Ausgabedaten, dass `catch`-Element, die jeweiligen Fault Handler, sowie das `compensationHandler`-Element, die zuverwendenten Kompensations-Aktivitäten.

receive, reply

Da BPEL4WS Prozesse nach außen als Web Services repräsentiert werden, müssen die von Prozessen angebotenen Schnittstellen-Operationen in geeigneter Form bedient werden. Hierfür werden `receive`- und `reply`- Aktivitäten verwendet. `receive`-Aktivitäten warten hierbei auf den Aufruf von Schnittstellen-Operationen, bei dessen Eintreten sie die übergebenen Eingabedaten, falls vorhanden, in einer Variable ablegen und beenden. Für den Fall der Bedienung von One-Way Schnittstellen-Operationen sind diese damit vollkommen ausreichend, wohingegen bei synchronen Schnittstellen-Operationen zusätzliche Rückgabewerte an den Aufrufer zurückgesendet werden müssen. Für diesen Fall existieren `reply`-Aktivitäten, die das korrespondierende Gegenstück zu `receive`-Aktivitäten darstellen, die die Rückgabewerte, die in Form von Variablen den `reply`-Aktivitäten übergeben werden, an den Aufrufer, der bis dahin blockiert, zurücksenden. Anhand des nachfolgenden Listings ist die Syntax von `receive`- und `reply`-Aktivitäten dargestellt:

```
<receive partnerLink="name" portType="name" operation="name"
         variable="name"    createInstance="yes|no">
    <correlations>
        <correlation set="name" initiate="yes|no"/>
    </correaltions>
</receive>

<reply partnerLink="name" portType="name"  operation="name"
         variable="name"?  faultName="name"?>
    <correlations>
        <correlation set="name" initiate="yes|no"/>
    </correaltions>
</receive>
```

Die `partnerLink`-Attribute charakterisieren die zu verwendenden Partner Links, die `portType`- und `operation`-Attribute, die zu bedienenden Schnittstellen-Operationen, die `variable`-Attribute jeweils die Variablen, die die Ein- bzw. die zu übergebenden Ausgabedaten halten und das `faultName`-Attribut den Typ einer als WSDL-Fehlernachricht zu übermittelnder Rückgabenachricht.

throw

`throw`-Aktivitäten werden innerhalb von BPEL4WS Prozessen verwendet um explizit Fehler zu signalisieren, die anhand eindeutiger Namen identifiziert und durch zusätzliche Informationen anhand von Fehler-Variablen genauer charakterisiert werden, auf die mittels Fault Handlern innerhalb von BPEL4WS Prozessen reagiert werden kann. Die Syntax von `throw`-Aktivitäten zeigt das nachfolgende Listing:

```
<throw faultName="name" faultVariable="name">
    standard-elements
</throw>
```

Das `faultName`-Attribut benennt den auszulösenden Fehler und das `faultVariable`-Attribut referenziert eine Variable, die zusätzliche Informationen über den Fehler enthält.

wait

`wait`-Aktivitäten ermöglichen es BPEL4WS Prozesse entweder eine bestimmte Zeitdauer oder bis zum Eintritt eines gewissen Zeitpunktes zu warten. Ihre Syntax ist im nachfolgenden Listing dargestellt:

```
<wait (for="duration-expr" | until="deadline-expr")>
    standard-elements
</wait>
```

Das `duration-expr`-Element spezifiziert hierbei einen Duration-Wert Ausdruck bzw. das `deadline-expr`-Element einen Deadline-Wert Ausdruck.

empty

Es gibt Situationen, in denen es notwendig ist „Nichts“ zu tun, beispielsweise bei der Behandlung von Fehlern um diese zu unterdrücken, hierfür existiert in BPEL4WS die `empty`-Aktivität, deren Syntax im nachfolgenden Listing dargestellt ist.

```
<empty>
</empty>
```

3.2.9 Struktur-Aktivitäten

Die bisher beschriebenen Aktivitäten haben lediglich die von Geschäftsprozessen durchzuführenden Arbeitsschritte, nicht jedoch deren Kombination zu Prozessen und damit verbunden die Spezifikation des Kontrollflusses, charakterisiert. Hierfür werden in BPEL4WS sog. Struktur-Aktivitäten verwendet, die wiederum mehrere Aktivitäten, sowohl Basis- als auch Struktur-Aktivitäten, zusammenfassen und deren Kontrollfluss spezifizieren.

Für die Spezifikation des Kontrollflusses anhand von Struktur-Aktivitäten wurden in BPEL4WS zwei unterschiedliche Ansätze, die jeweils im Kontext der BPEL4WS Ursprungssprachen XLANG [39] und WSFL [30] verwendet werden, miteinander kombiniert. Zum einen die Graph- bzw. Fluss-ähnliche Spezifikation von WSFL anhand von `flow`-Aktivitäten sowie die Programm- bzw. Block-ähnliche Spezifikation von XLANG anhand `sequence`-, `switch`-, `while`- und `pick`-Aktivitäten, die im Folgenden näher beschrieben werden.

sequence

`sequence`-Aktivitäten beschreiben die sequentielle Ausführung der von ihnen zusammengefassten Aktivitäten in der sie angegebenen Reihenfolge, die, sobald alle eingebetteten Aktivitäten ausgeführt wurden, beendet werden. Ihre Syntax ist im nachfolgenden Listing dargestellt:

```
<sequence>
    activity+
</sequence>
```

Hierbei charakterisiert das `activity`-Element eine oder mehrere Basis- bzw. Struktur-Aktivitäten.

switch

`switch`-Aktivitäten charakterisieren Mehrfachverzweigungen, deren einzelne Zweige anhand von `case`-Elementen spezifiziert werden. Ihre Syntax ist im nachfolgenden Listing dargestellt:

```
<switch>
     <case condition="bool-expr">+
         acitivity
     </case>
     <otherwise>?
         activity
     </otherwise>
</switch>
```

Jedes `case`-Element besitzt ein `condition`-Attribut, welches anhand eines boolschen Ausdrucks die Bedingung des Zweiges, der, wenn die Bedingung erfüllt ist, die Ausführung der durch `activity` spezifizierten Aktivität zur Folge hat, repräsentiert. Hierbei gilt, dass der erste Zweig, dessen Bedingung erfüllt ist, ausgeführt wird. Für den Fall, dass keine Bedingung erfüllt ist, kommt der durch das optionale `otherwise`-Element spezifizierte Zweig zur Ausführung. In Analogie zu `sequence`-Aktivitäten wird die Ausführung von `switch`-Aktivitäten beendet, sobald alle Aktivität des gewählten Zweiges beendet wurden.

while

`while`-Aktivitäten charakterisieren die wiederholte Ausführung einer durch `activity` spezifizierten Aktivität, deren Syntax im nachfolgenden Listing dargestellt wird:

```
<while condition="bool-expr">
    standard-elements
    activity
</while>
```

Die Wiederholung erfolgt hierbei solange, wie der durch das `condition`-Attribut spezifizierte boolsche Ausdruck erfüllt ist.

pick

`pick`-Aktivitäten erwarten das Eintreten eines Ereignisses entweder in Form einer empfangenen Nachricht, durch den Aufruf einer Prozess-Service-Operation, oder in Form des Überschreitens einer spezifizierten Zeitmarke bzw. dem Verstreichen eines Zeitintervalls, auf die durch den Aufruf einer ihr zugeordneten Aktivitäten reagiert wird. Aufgrund der nicht Vorhersagbarkeit des Eintretens von Ereignissen besitzen diese Aktivitäten einen nicht-deterministischen Charakter. Anhand des nachfolgenden Listings ist deren Syntax dargestellt:

```
<pick createInstance="yes|no">
    <onMessage partnerLink="name" portType="qname"
               operation="name"   variable="name">+
        <correlations>
            <correlation set="name" initiate="yes|no"/>
        </correlations>
        activity
    </onMessage>
    <onAlarm (for="duration-expr" | until="deadline-expr")>
        activity
    </onAlarm>
</pick>
```

Die `onMessage`-Elemente spezifizieren hierbei Ereignisse, die durch den Aufruf der durch `portType` und `operation` charakterisierten Prozess-Schnittstellen-Operation ausgelöst werden, auf die durch eine

anhand von `activity` spezifizierte Aktivität reagiert wird. Demgegenüber charakterisieren `onAlarm`-Elemente Ereignisse, die entweder beim Erreichen von Zeitpunkten (`deadline-expr`) oder dem Verstreichen von Zeitintervallen (`duration-expr`) ausgelöst werden, auf die durch eine anhand von `activity` spezifizierte Aktivität reagiert wird. Nachdem `pick`-Aktivitäten ein Ereigniss akzeptiert haben, werden weitere Ereignisse nicht mehr berücksichtig. Sie werden beendet, sobald die korrespondierende Aktivität zur Ereignissbehandlung beendet wurde.

flow

`flow`-Aktivitäten bieten die Möglichkeit mehrere Aktivitäten parallel auszuführen und, wenn notwendig, diese untereinander zu synchronisieren. Ihre Syntax zeigt das nachfolgende Listing:

```
<flow>
   <links>?
      <link name="name"/>+
   </links>
   activity+
</flow>
```

Hierbei spezifizieren `link`-Elemente, die im Kontext der `flow`-Aktivität verwendeten Synchronisationskanten, die jeweils zwischen einer Quell-Aktivität, die anhand des `source`-Elements mit Verweis auf einen Link `name`n und einer Ziel-Aktivität, die anhand des `target`-Elements mit Verweis auf den gleichen Link `name`n, festgelegt werden, deren Semantik die Ausführung der Quell-Aktivität vor der Ziel-Aktivität fordert. `source`- und `target`-Element sind dabei Standardelemente, deren Syntax im nachfolgenden Beispiel erkennbar ist, die sowohl bei Struktur- als auch Basis-Aktivität verwendet werden können.

```
<flow>
    <links>
        <link name="AtoB"/>
    </links>
    <invoke name="A" ...>
        <source linkName="AtoB" transitionCondition="..."/>
    </invoke>
    <invoke name="B" joinCondition="...">
        <target linkName="AtoB"/>
    </invoke>
</flow>
```

Dieses Beispiel spezifiziert eine Synchronisationskante `AtoB` zwischen der `invoke`-Aktivität `A` und `B`, aufgrund derer die Aktivität `B` erst ausgeführt werden kann sobald `A` beendet wurde.

Durch die zusätzliche Angabe von `transitionCondition`s auf Seiten der Quell- und `joinCondition`s auf Seiten der Ziel-Aktivitäten kann die Ausführung von Ziel-Aktivitäten im Kontext von `flow`-Aktivitäten beeinflusst werden, wodurch Links vergleichbar mit Kanten und Aktivitäten, vergleichbar mit Knoten, in einem Prozess-Fluss-Graphen sind. `transitionCondition`s charakterisieren hierbei, als boolscher Ausdruck, den Status von Synchronisationskanten, der nach Beendigung der zugeordneten Quell-Aktivität automatisch ermittelt wird, die zur Bestimmung der von Aktivitäten, die eingehenden Synchronisationskanten besitzen, explizit oder implizit spezifizierten `joinCondition`, ein boolscher Ausdruck über den Status aller eingehenden Synchronisationkanten, dient. Hierbei wird bei der Aktivierung von Aktivitäten, deren `joinCondition` zu falsch ausgewertet wurde, ein interner Fehler erzeugt, der, wenn das `suppressJoinFailure`-Attribut des `process`-Elements mit `true` spezifiziert wurde, unterdrückt bzw. durch eine `empty`-Aktivität behandelt wird, wodurch die jeweilige Aktivität nicht zur Ausführung gelangt. In diesem Fall werden gleichzeitig die `transitionConditions` aller ausgehenden Synchronisationskanten zu falsch evaluiert.

3.2.10 Scopes

Zusätzlich zu den gerade beschriebenen Konstrukten existieren in BPEL4WS Scopes, die logische Arbeitseinheiten von BPEL4WS Prozessen charakterisieren, mehrere Aktivitäten umfassen und diesen einen gemeinsamen Kontext in Form eigener Variablen, eigener Fehler- und Ereignisbehandlung sowie eigenen Korrelationsmengen zur Verfügung stellen, die lediglich im Kontext des jeweiligen Scope Gültigkeit besitzen. Scopes werden anhand von `scope`-Aktivitäten spezifiziert, deren Syntax im Folgenden dargestellt ist:

```
<scope>
    <variables>            ... </variables>?
    <correlationSets>      ... </correlationSets>?
    <faultHandlers>        ... </faultHandlers>?
    <compensationHandlers> ... </compensationHandlers>?
    <eventHandlers>        ... </eventHandlers>?

    activity
</scope>
```

Hierbei spezifiziert `activity` die Aktivität, die im Kontext von Scopes die auszuführenden Arbeitsschritte beschreibt, dass `variables`-Element, die in Scopes spezifizierten Variablen, dass `correlationSets`-Element, die im Kontext von Scopes verwendeten Korrelationsmengen, dass `faultHandlers`- und `compensationHandlers`-Element, die spezifizierten Fehler- und Kompensationsbehandlungen sowie das `eventHandlers`-Element, die verwendeten Ereignisbehandlungen. Dabei sind bis auf `activity` alle Elemente optional. An dieser Stelle sei noch bemerkt, das das `process`-Element ebenfalls ein Scope repräsentiert, dass gegenüber durch `scope`-Aktivitäten spezifizierte lokalen Scopes als globaler Scope angesehen werden kann.

Ereignisbehandlung

Die im Kontext von Scopes spezifizierbaren `eventHandlers` charakterisieren Ereignisbehandlungen, die bei Eintreten bestimmter Ereignisse parallel zum Scope ausgeführt werden. BPEL4WS unterscheidet hierbei im wesentlichen zwischen zwei Ereigniss-Arten: Nachrichten-Ereignisse, die beim Empfang einer spezifischen Nachricht ausgelöst werden und Timeout-Ereignisse, die beim Erreichen eines bestimmten Zeitpunktes oder nach Ablauf einer bestimmten Zeitdauer erzeugt werden. Das nachfolgende Listing zeigt die Syntax der Spezifikation von `eventHandlers`:

```
<eventHandlers>?
    <onMessage partnerLink="ncname" portType="qname"
               operation="ncname"   variable="ncname"?>*
        <correlations>?
            <correlation set="ncname" initiate="yes|no">+
        </correlations>
        activity
    </onMessage>

    <onAlarm (for="duration-expr" | until="deadline-expr")>*
        activity
    </onAlarm>
</eventHandlers>
```

Hierbei charakterisieren `onMessage`-Elemente Nachrichten-Ereignisse und `onAlarm`-Elemente Zeitereignisse, auf die jeweils mit einer durch `activity` charakterisierten Aktivität, bei deren Eintreten, reagiert wird.

3.2.11 Fehlerbehandlung

Da Geschäftsprozesse in der Regel langlebig und zeitintensiv sind und darüber hinaus hochgradig sensitive Daten verarbeiten, ist eine ausreichende Fehlerbehandlung zwingend notwendig, die in BPEL4WS anhand von FaultHandlern, mit denen auf Fehler, die entweder bei Aufruf von Web Service Operationen, explizit anhand von `throw`-Aktivitäten oder durch die BPEL4WS Ausführungsumgebung, als interne Fehler, ausgelöst werden, reagiert werden kann und CompensationHandler, mit denen im Fehler-Fall bereits ausgeführte Arbeitsschritte durch Kompensation zurückgesetzt werden können, realisiert wird. Hierbei beschränkt sich die Fehlerbehandlung von BPEL4WS lediglich auf logische Arbeitsschritte (Scope). Im Folgenden werden Fault- und CompensationHandler näher charakterisiert.

FaultHandler

Fault Handler beschreiben Fehlerbehandlungen, die im Kontext von Scopes spezifiziert, bei Eintreten von Fehlern innerhalb des Scopes aktiviert und diese durch vorher spezifizierte Aktivitäten behandeln. Hierbei gilt für die Fälle, in denen keine Faulthandler für die im Scope aufgetretenen Fehler spezifiziert wurden, dass alle Aktivitäten angehalten, der Scope fehlerhaft beendet und alle Fehler an den übergeordneten Scope weitergereicht werden, wohingegen bei spezifizierten Faulthandlern die jeweiligen Fehlerbehandlungs-Aktivitäten abgearbeitet und nach deren Beendigung der Scope normal beendet wird. Dies ist wichtig, da die Art der Beendigung von Scopes Auswirkung auf die Möglichkeit zu deren Kompensation hat. Die Spezifikation von FaultHandler erfolgt anhand von `faultHandlers`-Elementen, deren Syntax im nachfolgenden Listing dargestellt ist:

```
<faultHandlers>
    <catch faultName="name" faultVariable="name">*
        activity
    </catch>
    <catchAll>
        activity
    </catchAll>
</faultHandlers>
```

`catch`-Elemente charakterisieren hierbei, die anhand der durch `activity` spezifizierten Aktivität zu behandelnden Fehler miitels ihres Fehlernames (`faultName`) oder dem Typ der übergebenen Fehlervariable (`faultVariable`) und das `catchAll`-Element, eine Fehlerbehandlung für unspezifizierte Fehler.

CompensationHandler

Wenn Fehler in Geschäftsprozessen eintreten, ist es notwendig im Kontext deren Fehlerbehandlung bereits durchgeführte Arbeitsschritte rückgängig zu machen. Da vielen der durchgeführten Arbeitsschritte nicht rücksetztbar sind, können diese lediglich durch andere kompensiert werden. Im Kontext von BPEL4WS werden hierzu CompensationHandler definiert, die diese Kompensationsschritte in Form von Aktivitäten, die beim Auslösen einer Rücksetzung ausgeführt werden, charakterisieren. Wie schon bereits FaultHandlern können diese lediglich für logische Arbeitsschritte (Scopes) spezifiziert werden. Die Charakterisierung von CompensationHandlern erfolgt anhand des `compensationHandler`-Elements, dessen Syntax im nachfolgenden Listing dargestellt ist:

```
<compensationHandler>
    activity
</compensationHandler>
```

Wobei `activity` die zur Kompensation auszuführende Aktivität charakterisiert.

CompensationHandler eines Scopes werden im Fehlerfall anhand der `compensate`-Aktivität ausgelöst,

die entweder im Kontext eines Faulthandlers eines Scopes, welches das Scope einschließt, für welches der CompensationHandler aufgerufen werden soll, oder in einem CompensationHandler, der unmittelbar das Scope einschließt, für welches der CompensationHandler aufgerufen werden soll, spezifiziert ist und das jeweilige zu kompensierende Scope normal beendet wurde.

Kapitel 4

Business Process Definition Metamodel

Zur Zeit werden von der Industrie zahlreiche Sprachen zur Modellierung von Geschäftsprozessen mit unterschiedlichen Ausrichtungen eingesetzt. Daher hat sich die Object Management Group (OMG) zum Ziel gesetzt eine einheitliche Sprache, das Business Process Definition Metamodel (BPDM), zur Beschreibung von Geschäftsprozessen auf Basis der Model Driven Architecture (MDA) zu standardisieren, die eine konsistente Lösung zur Geschäftsprozessmodellierung als auch -implementierung bietet. Hierzu wurde von der OMG ein Request for Proposal [37] veröffentlicht, indem die wesentlichen Anforderungen an diese Sprache beschrieben sind.

Dieses Kapitel stellt den vorläufigen von IBM, Adaptive, Borland, Data Access Technologies, EDS, MEGA und 88 Solutions gemachten Vorschlag zur Spezifikation des BPDM vor, wobei im ersten Abschnitt die Model Driven Architecture sowie im abschließenden Abschnitt die Beziehung zwischen BPDM, BPEL4WS und MDA charakterisiert wird.

4.1 Model Driven Architecture

Die Model Driven Architecture [5] der OMG beschreibt einen Ansatz zur Entwicklung von IT Systemen, bei der die Spezifikation der Funktionalität und der Benutzung eines Systems von den Details seiner Realisierung, auf der Grundlage einer konkreten Hardware- und Softwareplattform, getrennt wird. Hierzu wird von der MDA eine Rahmenvorgehensweise definiert, die die folgendene Schritte umfasst: Spezifikation des Systems unabhängig von einer Plattform; Plattform-Spezifikation; Auswahl einer Plattform für die Realisierung eines Systems; Transformation der Systemspezifikation in eine plattformspezifische Spezifikation. Die primären Ziele, die damit verfolgt werden, sind Interoperabilität, Portabilität und Wiederverwendbarkeit entwickelter Systeme.

Auf Grund der Rahmenvorgehensweise unterscheidet die MDA zwischen drei verschiedenen Modellarten, die das System aus unterschiedlichen Perspektiven beschreiben: das Computation-Independent Model (CIM), das Platform-Independent Model (PIM) sowie das Platform-Specific Model (PSM), die mittels geeigneter Transformationen ineinander umgewandelt werden können. Hierbei ist das Ziel, die Funktionalität eines Systems mittels PIM Modellen zu beschreiben, aus denen anhand von Transformationen und unter Verwendung von Plattform-Modellen (PM) (z.B. für J2EE, .NET oder CORBA) PSM erzeugt werden, mittels derer der Code zur Umsetzung des Systems bzgl. einer spezifischen Plattform direkt und automatisch erzeugt werden kann. Diese Vorgehensweise bildet das zentrale Systemkonstruktionsparadigma der MDA.

Den wesentlichen Kern des MDA Ansatz bilden die folgenden OMG Standards: Die Unified Modeling Language, die Meta Object Facility (MOF) [36], das Common Warehouse Metamodel (CWM) sowie der XML Metadata Interchange (XMI) Standard. UML spielt hierbei die zentrale Rolle, da es eine Modellierungssprache ist, die zum einen ausreichende Sprachmechanismen besitzt, um von plattformspezifischen Details zu abstrahieren, als auch über einen Detailierungsgrad verfügt, der allgemein benötigte Konzepte zur Spezifikation von Softwaresystemen zur Verfügung stellt. Zusätzlich bietet UML anhand seines Profile-Mechanismus die Möglichkeit ohne Erweiterung der Sprache diese an verschiedene Anwendungsbereiche anzupassen. Daher wird es im Kontext der MDA zur Spezifikation von CIM, PIM und PSM von Systemen eingesetzt. Ein weiterer wesentlicher Aspekt ist, dass UML anhand eines MOF kompatiblen Metamodells eindeutig spezifiziert ist, wodurch die Möglichkeit besteht, mittels MOF Query/View/Transformation (QVT), welches sich zum Zeitpunkt dieser Arbeit noch in der Phase der Standardisierung befand, UML Modelle in andere MOF kompatible Modellierungssprachen abzubilden, speziell im Kontext der Transformation von PIM's nach PSM's. Hierbei ist es nicht nur wichtig Modelle in andere Modelle abzubilden, sondern diese auch zwischen verschiedenen Tools austauschen zu können. Hierfür wurde XMI als Standard spezifiziert, welcher auf Basis des zugrundeliegenden Metamodells Regeln zur Definition eines XML Schemas beschreibt, anhand dessen Modelle zwischen Softwarewerkzeugen in Form von XML ausgetauscht werden können. Als Erweiterung der Kernstandards sieht die MDA verschiedene UML Profile zur Beschreibung von Standard-Services auf PIM Ebene wie Transaktions-, Events-, Sicherheits- und Verzeichnisdienste vor. Abbildung 4.1 zeigt die wesentlichen Standards im Kontext der MDA.

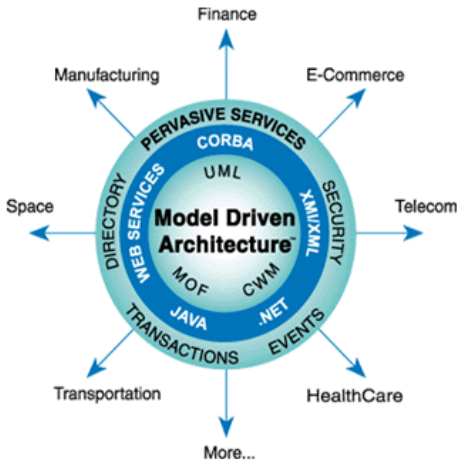

Abbildung 4.1: *OMG's Model Driven Architecture.*

Im Folgenden werden die im Kontext der MDA Verwendung findenden Modellarten näher charakterisiert.

4.1.1 Computation Independent Model

Das CIM beschreibt ein System in seiner zukünftigen Einsatzumgebung, speziell die Nutzung des Systems, seine Anforderungen an die Umgebung, sowie den Nutzen des Systems für die Umgebung. Es liefert prinzipiell keine Beschreibung der internen Struktur oder des internen Verhaltens des Systems sondern beschränkt sich sehr stark auf dessen externe Sicht. Darüber hinaus legt es den Grundwortschatz für die weiteren Modelle (PIM, PSM) im MDA Entwicklungsprozess fest.

Zur Repräsentation von CIM's mittels UML werden Anwendungsfalldiagramme (Use Case), Interaktionsdiagramme, Aktivitätsdiagramme als auch Klassendiagramme eingesetzt.

4.1.2 Platform Independent Model

Das PIM bildet das Kernstück des MDA Ansatzes. Es beschreibt die strukturellen und operationalen Aspekte eines Systems unabhänig von den später zur Implementierung einzusetzenden Hardware- und Softwareplattformen. Zur Realisierung einer solchen Plattformunabhängigkeit bieten sich klassische Verfahren, wie die Spezifikation auf Basis einer virtuellen Maschine oder die Nutzung einer generischen Plattform an. Hierbei können standardisierte Dienste und Elemente (pervasive Services, siehe Abbildung 4.1), die anhand von UML Profiles spezifiziert sind, ebenfalls Verwendung finden.

Durch die Tatsache, dass in einem PIM Buisness Logik, Domainwissen als auch Problemlösungen erfasst werden, die von plattformspezifischen Informationen getrennt sind, dienen sie als „ultimativer Know-How Speicher" für IT-Entwicklungsunternehmen, die zur Realisierung auf verschiedenen Plattformen zur Verfügung stehen und bei einem Technologiewechsel bleibende Gültigkeit besitzen [11].

Zur Darstellung von PIM's können alle in UML verfügbaren Modell- und Diagrammformen verwendet werden.

4.1.3 Platform Model

Plattform-Modelle dienen zur Spezifikation der Struktur und den Bestandteilen von Plattformen sowie deren angebotenen Diensten. Plattformmodelle können entweder generisch (z.B. Distributed Object Platform), technologiespezifisch (z.B. CORBA, J2EE) oder herstellerspezifisch (J2EE Realisierung im WebSphere Application Server) [11] sein. Die Modellierung und Einbindung erfolgt im Wesentlichen durch spezifische auf die Plattform angepasste UML Profile (z.B. CORBA UML Profile). Darüber hinaus beschreiben sie spezifische Anforderungen an die Verbindung der Plattform sowie die Verknüpfung von Anwendung und Plattform. Im Gegensatz zu PIM's ist ihre Gültigkeit und Lebensdauer an die beschriebene Plattform gebunden.

4.1.4 Platform Specific Model

In einem plattformspezifischen Modell wird ein PIM anhand konkreter Details über die Hard- und Softwareplattform, die anhand eines PM spezifiziert ist, ergänzt. Grundsätzlich kann ein PSM alle Details enthalten, die zur Implementierung des Systems notwendig sind. Andererseit besteht die Möglichkeit, dass ein PSM von einer Reihe von Implementationsdetails abstrahiert und Ausgangspunkt für weitere Transformationen ist (z.B. ein Modell, welches plattformspezifische Details bzgl. der J2EE Plattform enthält, kann als PIM gegenüber einem Modell betrachtet werden, welches zusätzlich herstellerspezifische Informationen über einen konkreten J2EE Application Server spezifiziert, auf welchem das System implementiert werden soll). Daher sind PIM und PSM immer als relativ zu betrachten.

Wie bereits beim PIM können beim PSM alle in UML verfügbaren Modell- und Diagrammformen verwendet werden. Speziell im Kontext von PSM, die alle Implementationsdetails enthalten, kommen verstärkt Deployment-Modelle zur Anwendung.

4.1.5 Transformation

In einem der vorherigen Abschnitte wurde erwähnt, dass das zentrale Systemkonstruktionsparadigma der MDA die Transformation abstrakter in detailiertere, näher an einer Hard- und Softwareplattform liegende Modelle ist. Daher sollen in diesem Abschnitt einige wesentliche Aspekte der Transformation anhand des PIM zu PSM Mappings beleuchtet werden. Hierbei müssen in den meisten Fällen neben der Spezifikation der Abbildungsregeln (Transformationsregeln) auch zusätzliche Informationen beispielsweise in Form von Plattformmodellen angegeben werden. Abbildung 4.2 zeigt das allgemeine Prinzip einer solchen Transformation.

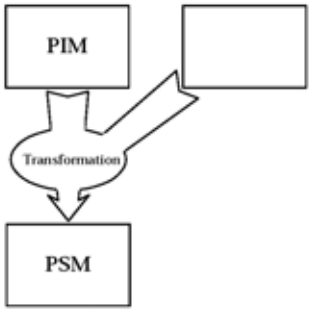

Abbildung 4.2: *Allgemeines Muster der Modelltransformation.*

Grundsätzlich lassen sich Modelltransformationen anhand der folgenden Kriterien klassifizieren: den Typ, der Form der Abbildungsregeln sowie der Transformationsstufe.

Hierbei beschreibt die Transformationsstufe, zwischen welchen Modellarten die Transformation durchgeführt wird:

- von PIM zu PIM, angewendet zur Verfeinerung von Modellen im Kontext iterativer Entwicklung.
- von PIM zu PSM, zur Umsetzung des Modells in einer ausführbaren Umgebung.
- von PSM zu PSM, speziell zur Komponentenrealisierung und zum Deployment.
- von PSM zu PIM, zur Abstraktion von Modellen existierender Implementierung.

Ein weiteres Kriterium ist die Art und Weise der Formulierung von Abbildungsregeln. Diese kann entweder in einem imperativen Stil erfolgen und beispielsweise anhand einer Programmiersprache wie Python oder Java implementiert werden [11] oder in Form von Aussagen (Bedingungen) über Quell- und Zielmodell der Transformation z.B. mittels Prädikatenlogik I Stufe. Die folgende Formel zeigt ein einfaches Beispiel einer solchen prädikatenlogischen Aussage: Sei $\mathcal{M}$ die Menge der Elemente des PIM und $\mathcal{Q}$ die Menge der Elemente des PSM, dann wird festgelegt, dass jedes Element des PIM auf mindestens ein Element des PSM anhand der Abbildungsfunktion f gemappt werden muss.

$$\forall x : x \in \mathcal{M} \rightarrow \exists y : y \in \mathcal{Q} \wedge f(x) = y$$

Hinzu kommt die Möglichkeit anhand des Typs Transformationen zu unterscheiden. Hierbei existieren im Wesentlichen drei verschiedene Typen: instanzbasierte, typbasierte und musterbasierte Transformationen.

Die typbasierte Transformationsform definiert Abbildungsregeln auf der Grundlage verschiedener Typen, die innerhalb des PIM und des PSM zur Modellierung verwendet werden. Damit wird es möglich

Abbildungsregeln unabhängig vom konkreten Modell, nur auf Grundlage der Modellierungselemente, zu spezifizieren. Ein Spezialfall der typbasierten Transformation ist die Metamodell Transformation, bei der die Typen beider Modelle durch MOF Metamodelle beschrieben werden. In Abbildung 4.3 ist eine solche Metamodell-Transformation am Beispiel der PIM zu PSM Transformation dargestellt.

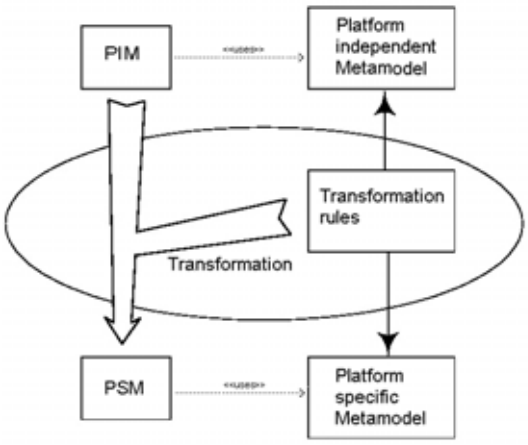

Abbildung 4.3: *Schematische Darstellung des Prinzips der typbasierten Transformation.*

Eine weitere Form ist die instanzbasierte Transformation. Bei dieser werden die Modellelemente im PIM, die in ein PSM überführt werden sollen, indentifiziert und plattformspezifisch markiert. Hierbei beschreiben die jeweiligen Markierungen, wie die markierten Elemente des PIM in spezifische PSM Elemente transformiert werden. Da die Markierungen nicht Bestandteil des PIMs sind, wird in diesem Kontext auch von einer zusätzlichen transparenten Markierungs-Schicht gesprochen, die über das PIM gelegt wird. Abbildung 4.4 zeigt das Prinzip einer solchen Transformation.

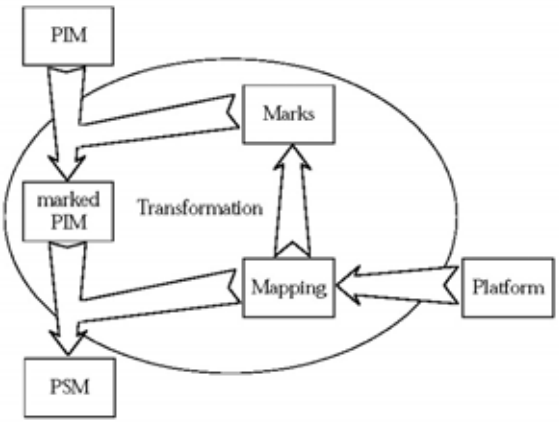

Abbildung 4.4: *Schematische Darstellung des Prinzips der instanzbasierten Transformation.*

Grundsätzlich besteht bei instanzbasierten Transformationen das Problem, dass die verwendeten Markierungen modellspezifisch sind. Daher werden in den Fällen, wo ähnliche Markierungen häufig und in unterschiedlichen Modellen anzubringen sind, diese zu Markierungsmustern oder -Templates zusam-

mengefasst. Daher bezeichnet man diese Form als musterbasierte Transformation.

4.2 Business Process Definition Metamodel

Das Business Process Definition Metamodel definiert eine abstrakte, Plattform unabhängige Sprache zur Spezifikation innerhalb eines Unternehmens ausführbarer Geschäftsprozesse (mit oder ohne menschlichen Einfluss), die zum einen selbständig ablaufen oder mit anderen unabhängigen, über Unternehmensgrenzen hinweg verteilten Geschäftsprozessen interagieren [37]. Sie befindet sich zur Zeit im Prozess der Standardisierung und soll im wesentlichen die folgenden Anforderungen erfüllen [37]:

- eine Vereinheitlichung der in der Industrie verwendeten Sprachen zur Beschreibung von Geschäftsprozessen ermöglichen.
- ein Metamodell spezifizieren, welches die existierenden UML Metamodelle um die Fähigkeit der Spezifikation von Geschäftsprozessen ergänzt.
- die Fähigkeit besitzen, Prozessmodelle für Workflow Management Prozesse, automatisierte Geschäftsprozesse und Kollaborationen zwischen Unternehmenseinheiten zu integrieren.
- die Möglichkeit zur Verfügung stellen, Geschäftsprozessmodelle zwischen Tools aufgrund von XMI auszutauschen.
- die Fähigkeit bieten, Kollaborationen zwischen den beteiligten Business Entities zu beschreiben, indem leichtgewichtige Kollaborationsmechanismen (wie Web Services) verwendet werden (Choreographie) und dies in Einklang mit der Möglichkeit der Definition interner (sub) Geschäftsprozesse bringen.

Im Folgenden soll der vorläufige Vorschlag [21] von IBM, Adaptive, Borland, Data Access Technologies, EDS, MEGA und 88 Solutions dargestellt werden, der sich zur Zeit im Prozess der Entwicklung befindet und der, mit Stand der aktualisierten Submission vom August 2004 [21], Gegenstand weiterer Betrachtungen sein wird.

Im Zuge dieses Vorschlags werden die wesentlichen, für die Geschäftsprozessmodellierung notwendigen, Konzepte ermittelt, die aufgrund der Tatsache, dass UML 2.0 bereits viele der zur Modellierung notwendigen Elemente enthält und die Sprache im Kontext der MDA verwendbar sein soll, auf das UML Metamodell abgebildet und in einem UML Profile, dem Business Process Definition Metamodel, zusammengefaßt werden. Hierbei werden die vom BPDM unterstützten Konzepte anhand eines konzeptuellen Metamodells, welches den Vorschlag zum gegebenen Zeitpunkt charakterisiert und in UML beschrieben ist, festgelegt. Die folgende Abbildung zeigt einen Ausschnittdes konzeptuellen Metamodells, anhand dessen einige wesentlichen Grundkonzepte erkennbar sind.

Ein Prozess ist ein „Container“ der seinen Zustand und sein Verhalten nach aussen kapselt, indem er über sichtbare Schnittstellen (*Interface*) verfügt, über die er anhand von Nachrichten (*Message*) mit anderen Prozessen interagiert und sein Verhalten durch eine Menge beinhaltender *Steps*, die beispielsweise das Versenden von Nachrichten auslösen, realisiert. Das hierbei zur Anwendung kommende Kommunikationsmodell basiert auf der Übertragung von Nachrichten. Hierzu werden zwischen den einzelnen Prozessen bzw. Partnern Nachrichten ausgetauscht, auf die auf Empfängerseite anhand spezieller Tasks innerhalb des Prozesses bzw. auf Seiten des Partners reagiert werden kann.

Diese Konzepte verdeutlichen, dass im BPDM das Prinzip der SOA auf den Bereich der Geschäftsprozessmodellierung übertragen wird, indem Prozesse als Komponenten verstanden werden, die nach aussen über klar definierte Schnittstellen verfügen, ihr Verhalten komplett kapseln und mit anderen

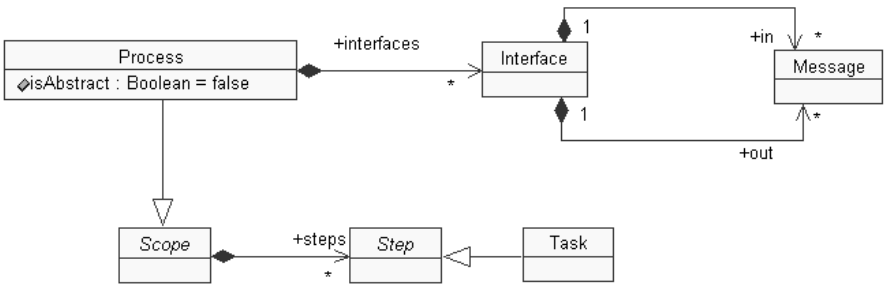

Abbildung 4.5: *Ausschnitt aus dem konzeptuellen Metamodell des BPDM.*

Prozessen komponentenartig zu einer Gesamtlösung kombiniert werden können. Daher kann prinzipiell bei der Modellierung von Geschäftsprozessen im BPDM zwischen einer externen und internen Sicht unterschieden werden.

Die durch das konzeptuelle Metamodell festgelegten Konzepte werden, wie bereits in einem der vorherigen Absätze erwähnt, anhand der Abbildung auf Elemente des UML 2.0 Metamodells [34] realisiert, die im wesentlichen innerhalb der folgenden UML Packages spezifiziert sind:

- `UML::Activities::(*)`[1]
- `UML::Actions::(*)`
- `UML::AuciliaryConstructs::(PrimitiveTypes, Templates)`
- `UML::Classes:(Dependencies, Interfaces, Kernel)`
- `UML::CommonBehaviors::(BasicBehavior, Communications, Time)`
- `UML::CompositeStructure::(Collaborations, InternalStructures, Ports, Structured-Activities, StructuredClasses)`
- `UML::StateMachines::(BehaviorStateMachines, ProtocolStateMachines)`
- `UML::UseCases`

Da einige Konzepte nicht direkt auf UML 2.0 abbildbar sind, wurden bereits zusätzliche Stereotypen definiert.

Aufgrund der bisher unvollständigen Spezifikation des BPDM wird im Folgenden dieses anhand des konzeptuellen Metamodells näher charakterisiert. Hierbei werden zum einen die jeweiligen Elemente zur Beschreibung der externen und internen Sicht eines Geschäftsprozesses, sowie, wenn bereits spezifiziert, ihre Realisierung mittels Elementen von UML 2.0 dargestellt. Die graphische Notation der jeweiligen konzeptuellen Metamodellelemente wird durch die bisher festgelegten korrespondierenden UML Elemente spezifiziert. Damit zwischen den einzelnen Elementen besser unterschieden werden kann, werden die Elemente des konzeptuellen Metamodells in *kursiver* und die des BPDM (UML-Metamodell Elemente) in `nicht-proportionaler` Schrift dargestellt.

[1] * charakterisiert dass ebenfalls alle Subpackages enthalten sind

4.2.1 Externe Sicht

Die externe Sicht eines Geschäftsprozesses beschreibt die Details, die von einem Prozess-Designer benötigt werden, um diesen in sein Prozessmodell zu integrieren. Sie spezifiziert die Kollaborationen, die zwischen einem Prozess und seinen Partnern stattfinden sowie die damit verbunden notwendigen Schnittstellen auf Seiten des Prozesses und der Partner. Die folgende Abbildung zeigt einen Ausschnitt aus dem konzeptuellen Metamodell des BPDM, in dem die Elemente zur Definition der externen Sicht spezifiziert sind.

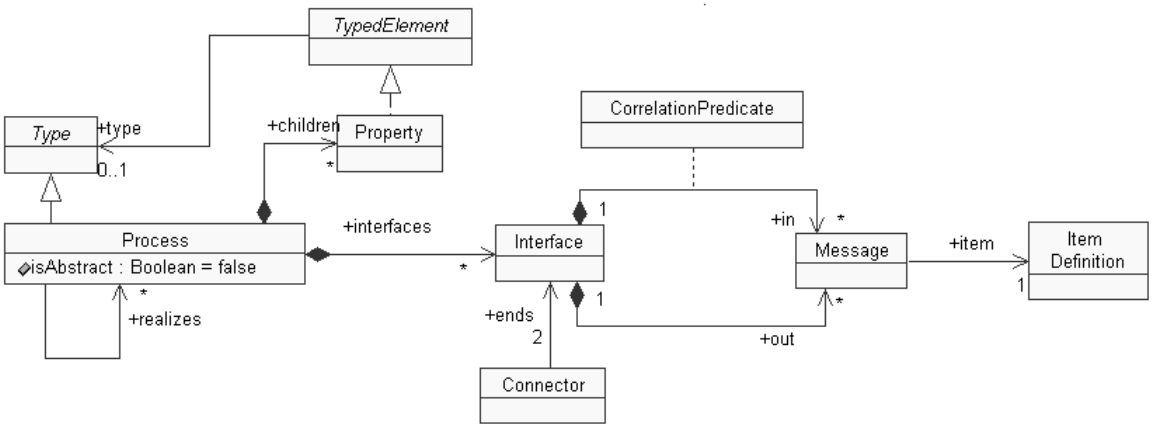

Abbildung 4.6: *Ausschnitt aus dem konzeptuellen Metamodell, durch das die Elemente zur Beschreibung der externen Sicht spezifiziert sind.*

Prozess

Ein Prozess stellt rein konzeptuell einen „Container“ dar, der zum einen sein Verhalten in Form von durch einen gerichteten Graphen verbundene *Steps* kapselt, sowie nach aussen über sichtbare Schnittstellen verfügt, über die er mit anderen Prozessen bzw. Partnern über den Austausch von Nachrichten interagiert.

Es wird zwischen zwei unterschiedlichen Arten von Prozessen anhand des *isAbstract*-Attributs unterschieden: abstrakte und reale (operationale) Prozesse. Abstrakte Prozesse sind vergleichbar mit abstrakten Klassen der objektorientierten Modellierung die lediglich den Rumpf und damit das Verhalten des Prozesses nach aussen beschreiben und von einem oder mehreren realen Prozessen „realisiert“ werden. Die Kopplung von abstrakten und realen Prozessen erfolgt anhand der *realize*-Assoziation, deren Semantik bisher nicht weiter spezifiziert wurde. Abbildung 4.7 zeigt die graphische Notation von Prozessen bzgl. ihrer externen Sicht sowie die Verknüpfung zwischen abstrakten und operationalen Prozessen.

Darüberhinaus können Prozesse ein oder mehrere Sub- bzw. Kind-Prozesse besitzen, die einen Teil des Verhaltens des Prozesses realisieren. Subprozesse stellen hierbei normale Prozesse dar, die bezüglich dieses Prozesses (Vater-Prozess) als Subprozesse verwendet werden und anhand der *children*-Assoziation dem Prozess als eine durch einen Prozess getypte *Property* zugeordnet werden. Aufgrund der Prozess-Typisierung des *Property* Elements stellt die Zuordnung eine Referenz-Zuordnung dar, wodurch Subprozesse in anderen Prozessmodellen als normale Partner-Prozesse verwendet werden können. Prinzipiell sind Subprozesse nicht als Prozess-Komponenten zu verstehen, aus denen der

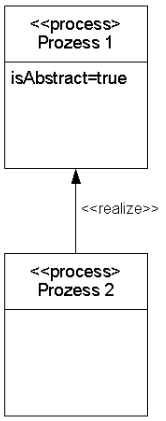

Abbildung 4.7: *Graphische Notation der externen Sicht von Prozessen sowie der Verknüpfung zwischen abstrakten und operationalen Prozessen.*

Prozess komplett aufgebaut ist, sondern deren Lebenszyklus lediglich von dem des Vater-Prozesses abhängig ist. D.h. wird die Vater-Prozessinstanz beendet, so werden alle Kind-Prozessinstanzen beendet. Die Interaktion zwischen Vater- und Kind-Prozessen verläuft in gleicher Weise wie zwischen normalen Prozessen, anhand des Austauschs von Nachrichten über *Interfaces*. Die graphische Darstellung von Subprozessen zeigt Abbildung 4.8.

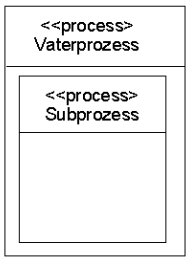

Abbildung 4.8: *Notation von Subprozessen.*

Dieses Prozess-Konzept wird im BPDM anhand einer durch den Stereotyp `process` versehenen Klasse (Prozess-Klasse) umgesetzt, die zum einen anhand von `Ports` die Schnittstellen spezifiziert, die der Prozess seinen Partnern zur Verfügung stellt, als auch die vom Prozess auf Seiten der Partnern benötigt werden. Das interne Verhalten wird demgegenüber in Form einer der Klasse über die `classifierBehavior` UML Metamodell-Assoziation zugeordneten Aktivität (siehe Abschnitt 4.2.2) beschrieben. Darüberhinaus werden die vom Prozess verwendeten Subprozesse ebenfalls als normale Prozess-Klassen modelliert, die über `Property`s, mit Kompositionssemantik, dem Prozess zugeordnet werden. Die hierfür verwendeten Modellierungskonstrukte sind durch das `UML::CompositeStructure::(StructuredClasses, Ports)` sowie das `UML::CommonBehaviors::BasicBehavior` UML Package im BPDM spezifiziert.

Prozess-Schnittstellen

Wie bereits im vorherigen Abschnitt erwähnt, verfügen Prozesse über Schnittstellen (*Interface*), anhand derer die Interaktion mit der Umwelt, d.h. mit anderen Prozessen bzw. Partnern, stattfindet.

Hierbei bündelt eine Schnittstelle verschiedene Nachrichten (*Messages*), die entweder von einem Prozess an einen seiner Partner (über *out* assoziiert) oder durch einen Partner an den Prozess (über *in* assoziiert) versendet werden. *out-Messages* repräsentieren dabei Anforderungen an einen Partner, der bezüglich der Interaktion mit diesem Prozess diese Nachricht verarbeiten muss. *in-Message*s hingegen Nachrichten auf die der Prozess reagiert. Jede *Message* dient als Transport-Element für ein durch eine *Item Definition* spezifizierte Entität. Die Notation von *Interface*s und *Message*s wird anhand der nachfolgenden Abbildung gezeigt.

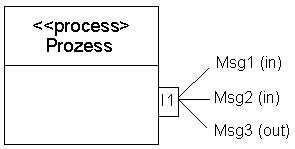

Abbildung 4.9: *Notation von Interfaces und Messages.*

Die Umsetzung von *Interface* und *Message* im BPDM erfolgt anhand von `Port`, `Interface`, `Operation` und `Reception`. Hierbei wird ein *Interface* anhand eines `Interface` realisiert, welches einen der Prozess-Klasse zugeordneten `Port` typisiert. Die jeweiligen durch ein *Interface* spezifizierten möglichen eingehenden Nachrichten werden in Form von Interface-`Operationen` abgebildet, die zusätzlich mit einer `Reception` versehen sind.

Partner-Rollen und Kollaborationen

Prozesse interagieren zu unterschiedlichen Zeitpunkten mit verschiedenen Partnern um das durch den Prozess spezifizierte Verhalten zu realisieren. Daher ist es notwendig, die Interaktionen zwischen Prozessen und Partnern im Modell in geeigneter Art und Weise zu spezifizieren. Hierzu werden Partner-Rollen und Kollaborationen verwendet.

Die mit einem Prozess interagierenden Partnern werden in Form von Partner-Rollen beschrieben, die lediglich die Anforderungen, die von diesen Partnern, die diese Rolle inne haben, erfüllt werden, charakterisieren. Sie sind die Basis-Elemente zur Beschreibung von Interaktionen, indem unabhängig von einem konkreten Prozess oder Partner die Interaktionen in Form interagierender Partner-Rollen spezifiziert werden. Rein konzeptuell werden sie anhand von abstrakten Prozessen beschrieben, deren *Interface*s die Nachrichten spezifizieren, die der die Rolle inne habende Partner versenden und empfangen kann.

Die Interaktionen zwischen Partnern werden anhand von Kollaborationen zwischen Partner-Rollen beschrieben, die das Interaktionsmuster anhand auszutauschender Nachrichten vom Standpunkt eines externen Betrachters spezifizieren. Dies hat den Vorteil, dass Kollaborationen im Kontext anderer Szenarien mit anderen Partnern, die nach dem selben Muster interagieren, wiederverwendet werden können. Die Kopplung einer Kollaboration mit den jeweiligen Partnern erfolgt anhand der Zuordnung der Partner zu den sie einnehmenden Rollen, indem die, die Partner repräsentierenden, operationalen Prozesse mit den jeweiligen abstrakten Prozessen anhand der *realize*-Assozation verknüpft werden. Rein konzeptuell werden Kollaborationen anhand von über *Connector*s verbundene abstrakte Prozesse, die die Rollen spezifizieren, beschrieben. Die nachfolgende Abbildung zeigt anhand der Spezifikation der Bestell-Kollaboration des Bestell-Prozesses eine mögliche graphische Notation von Rollen und Kollaborationen. Hierbei sind Rollen als gestrichelte Rechtecke symbolisiert, Kollaborationen als die Rollen verbindende Linie, an der die im Kontext der Kollaboration ausgetauschten Nachrichten

spezifiziert sind.

Abbildung 4.10: *Beispiel der Spezifikation einer Kollaboration.*

Im BPDM werden die jeweiligen Rollen ebenfalls mittels Prozess-Klassen, die abstrakte Prozesse darstellen, beschrieben, wobei die zwischen den Rollen existierenden Kollaborationen anhand von `Collaboration`s spezifiziert werden, die mehrere `Interface`s mittels `Connector`s verbinden. Die Übertragung der Kollaborationsspezifikation auf konkrete Partner (andere Prozessklassen), zwischen denen diese abläuft und die jeweils eine der definierten Rollen inne haben, erfolgt anhand von `CollaborationOccurence`.

Kollaborationsprotokolle

Kollaborationsprotokolle (Geschäftsprotokolle) beschreiben die Art und Weise des Ablaufs der von Kollaborationen spezifizierten Interaktionen. Sie können konzeptuell entweder anhand von abstrakten Prozessen, die jeweils das von außen sichtbare Verhalten der an einer Kollaboration teilnehmenden Rollen beschreiben, oder mittels kollaborativen Prozessen, die den Ablauf von Kollaborationen aus Sicht eines unabhängigen Betrachters (aus globalem Blickpunkt) spezifizieren, beschrieben werden.

Abstrakten Prozessen sind Rollen in Kollaborationen zugeordnet bzw. sie entsprechen den Rollen, die die Struktur des Ablaufs der Kollaboration aus Sicht der jeweiligen Rolle, anhand abstrakter BPDM Prozesse spezifizieren. Als Beispiel zeigt Abbildung 4.11 die abstrakten Prozesse für die Verkäufer- und die Käufer-Rolle der Bestell-Kollaboration. Zu Beginn wird beim Verkäufer Prozess auf den Empfang einer Bestellung gewartet und zum Abschluss eine Bestell- und Versandbestätigung versendet. Der Käufer-Prozess demgegenüber versendet zu Beginn eine Bestellung und wartet auf eine Verstandbestätigung. Jeder Partner der eine dieser Rollen in der Bestell-Kollaboration inne hat, muss den jeweiligen abstrakten Prozess realisieren.

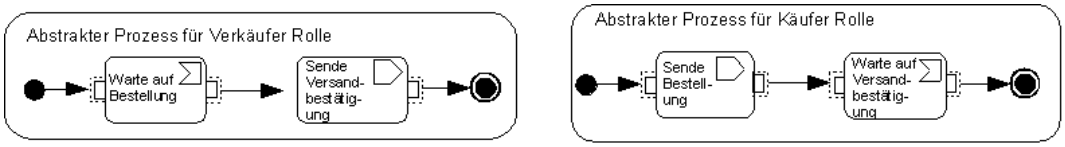

Abbildung 4.11: *Abstrakte Prozesse für die Käufer- und Verkäufer-Rolle.*

Kollaborative Prozesse beschreiben das Interaktionsprotokoll einer Kollaboration aus globaler Sicht [16] anhand eines Prozess-Graphen, der die Phasen der Interaktion mittels über gerichtete Kanten verbundene Interaktionsknoten, die die Interaktionen zwischen zwei Rollen von beiden Seiten (symmetrisch) beschreiben, spezifiziert. Zusätzlich wird in [16] postuliert, dass das durch einen kollaborativen Prozess beschriebene Verhalten einer Menge interagierender Rollen konsistent mit jedem für eine dieser Rollen spezifizierten abstrakten Prozess sein muss und damit aus abstrakten- der kollaborative- bzw. zu kollaborativen- die korrespondierenden abstrakten Prozesse erzeugt werden können. Bisher ist diese Abbildung speziell noch nicht weiter spezifiziert. Abbildung 4.12 zeigt eine mögliche Darstellung

des kollaborativen Prozesses für die Bestell-Kollaboration.

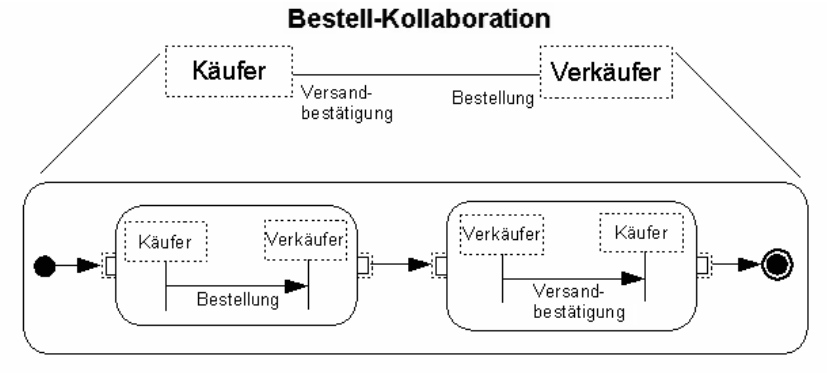

Abbildung 4.12: *Kollaborativer Prozess für die Bestell-Kollaboration.*

Bisher sind kollaborative Prozesse im konzeptuellen Metamodell nicht weiter spezifiziert. In Abbildung 4.12 ist jedoch ein kollaborativen Prozesses in einer möglichen Notation dargestellt.

4.2.2 Interne Sicht, Modellierung von Prozessverhalten

Die interne Sicht von Geschäftsprozessen beschreibt, wie das Verhalten von Prozessen realisiert wird. Es wird anhand einer geordneten Menge von *Step*s (Arbeitsschritten), die über Ein- und Austrittspunkte verfügen, die untereinander anhand von gerichteten Kanten verbunden sind und über die Daten in Form von Token zwischen den einzelnen *Step*s ausgetauscht und zur Aktivierung von ihnen konsumiert werden, modelliert. Abbildung 4.14 zeigt einen Ausschnitt aus dem konzeptuellen Metamodell, in dem einige Elemente zur Beschreibung des Prozessverhaltens spezifiziert sind.

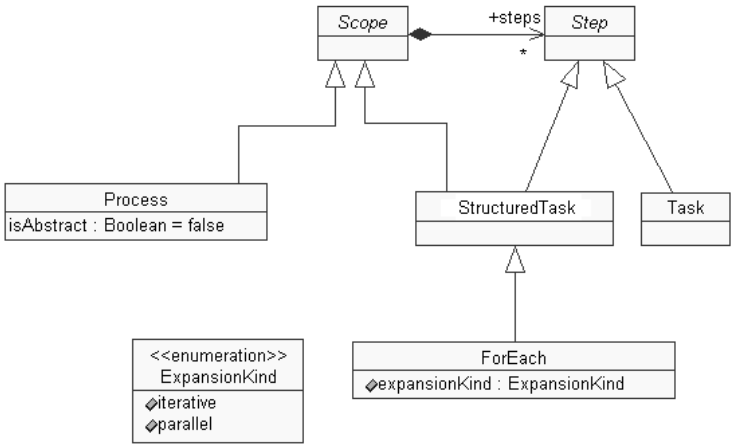

Abbildung 4.13: *Ausschnitt aus dem konzeptuellen Metamodell, der die Elemente zur Beschreibung der internen Sicht spezifiziert.*

Die Umsetzung dieser Konzepte erfolgt anhand der im `UML::Activities` Package spezifizierten Akti-

vitätsdiagrammen in Form von Aktivitäten (`Activity`), die den Kontroll- und Datenfluss als auch die vom Geschäftsprozess durchzuführenden Arbeitsschritte mittels unterschiedlicher Arten von Knoten, die über Kanten verbunden sind, anhand eines engebetteten Graphen spezifizieren. Hierbei kommen analog zur konzeptuellen Betrachtung Token zur Anwendung, die von Kanten zwischen Knoten weitergeleitet werden und die für die Ausführung einzelner Knoten sowie den Datentransport verantwortlich sind. Daher besitzen Aktivitätsdiagramme sehr große Ähnlichkeiten mit Petri-Netzen (Abschnitt 2.2.1), wodurch sie sogar als spezifischer Netz-Typ aufgefaßt werden können. Der Begriff Token ist dabei im Kontext von UML 2.0 wie folgt definiert [34]:

> *A token contains an object, datum, or locus of control, and is present in the activity diagram at a particular node. Each token is distinct from any other, even if it contains the same value as another.*

Die nachfolgende Abbildung zeigt als Beispiel die interne Sicht des in Abschnitt 2.1 dargestellten Geschäftsprozesses modelliert mit Hilfe des UML Aktivitätsdiagramms. Die wesentlichen Elemente werden in den nachfolgenden Abschnitten näher erläutert.

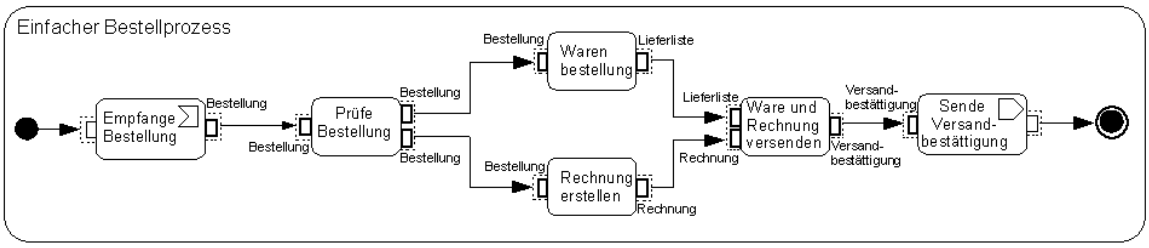

Abbildung 4.14: *Spezifikation der internen Sicht des Bestell-Prozesses.*

Steps

Steps stellen die Arbeitsschritte dar, die ein Prozess durchführt. Für die Modellierung werden im konzeptuellen Metamodell zwei unterschiedliche Arten von *Steps* unterschieden: *Tasks* und strukturierte Tasks (*StructuredTask*).

Tasks beschreiben die konzeptuellen Basiselemente zur Modellierung des Verhaltens von Geschäftsprozessen, die nicht weiter zerlegbar sind und atomare Arbeitsschritte darstellen. Sie können entweder manuell, durch Personen, oder durch Systeme realisiert werden, was anhand zugeordneter *Ressource-Requirements* spezifiziert wird (siehe hierzu Abschnitt 4.2.8). Demgegenüber repräsentieren strukturierte Tasks Tasks, die einen oder mehrere *Steps* zusammenfassen und nicht als eigenständige Prozesse sondern nur innerhalb des Prozesses verwendet werden können. Zusätzlich bilden sie einen lokalen Gültigkeitsbereich für beispielsweise *Variablen*.

Tasks werden in ihrer Umsetzung in Aktivitätsdiagrammen als ausführbare Knoten in Form von `Actions`, symbolisiert als Rechtecke mit runden Ecken (Abbildung 4.15), realisiert, die aufgrund ihres Typs verschiedene Verhalten besitzen können. Demgegenüber werden strukturierte Tasks als Aktivitäten (`Activity`) repräsentiert die anhand einer spezifischen `Action`, der `CallBehaviorAction`, aufgerufen werden und als Platzhalter für „eingebettete Aktivitäten“ angesehen werden können, wodurch gleiche Aktivitäten mehrmals innerhalb eines Prozesses verwendet werden können. Sie werden ebenfalls wie Tasks symbolisiert, jedoch enthält das Rechteck die jeweiligen vom strukturiertem Task zusammengefaßten *Steps*. Die Verbindung zwischen enthaltenen *Step*s und strukturierten Task erfolgt

über Ein- und Austrittspunkte (Abbildung 4.15).

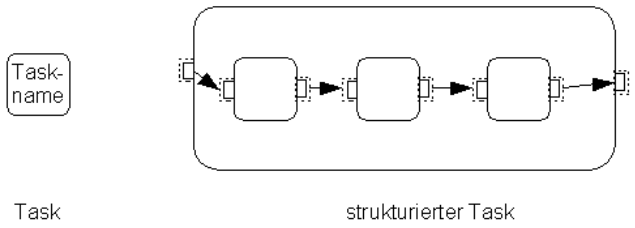

Abbildung 4.15: *Notation von Tasks und strukturierten Tasks.*

Eintritts- und Austrittspunkte von Steps

Steps besitzen einen oder mehrere Eintritts- und Austrittspunkte in Form von *Input-* und *Output-*Pins, anhand derer Eingaben und Ausgaben, in Form von Token, über gerichtete Kanten, die die *Pins* verbinden, dem Step über genau ein *Pin* (implizite ODER Semantik von Pins) zu- bzw. vom *Step* abgeführt werden. Zusätzlich können mehrere *Input-* bzw. *Output-*Pins untereinander gruppiert werden, wodurch gleichzeitig über alle *Pins* der Gruppe Eingaben bzw. Ausgaben eingehen bzw. ausgehen müssen (implizite UND Semantik gruppierter Pins). Darüber hinaus existieren spezielle *Output-*Pins, die im Kontext der Fehlerbehandlung eine wichtige Rolle spielen, siehe hierzu Abschnitt 4.2.5.

Konzeptuell wird zwischen zwei unterschiedlichen Arten von *Pins* unterschieden: Kontroll- und Daten-Pins. Kontroll-Pins können lediglich sog. Kontroll-Token aufnehmen bzw. emittieren, wohingegen Daten-Pins jegliche Art von typisierten Daten in Form von Token aufnehmen bzw. emittieren können. Der Typ von Daten-Pins ist anhand von Item Definitions festgelegt. Sie dienen im wesentlichen dazu den Kontroll- und Datenfluss eines Prozesses unabhängig voneinander spezifizieren zu können.

Dieses Konzept wird anhand von `Action`s bzw. `Activity`s zugeordneten `Pin`s (`InputPin`, `OutputPin`) bzw. `ActivityParameterNode` realisiert, die zusätzlich von `ParameterSets` gekapselt werden um die alternative Selektion-Semantik von *Pins* zu ermöglichen. Sie werden als den `Action`s zugeordnete Rechtecke symbolisiert, die, wenn sie untereinander gruppiert sind, durch ein gestricheltes Rechteck umschlossen sind. Die Unterscheidung zwischen Daten- und Kontroll-Pins erfolgt anhand der Randdicke des symbolisierten Rechtecks, wobei Kontroll-Pins durch einen dünneren Rand als Daten-Pins charakterisiert werden (siehe Abbildung 4.16).

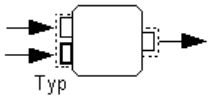

Abbildung 4.16: *Notation von gruppierten und ungruppierten Pins.*

Arten von Tasks

Rein konzeptuell können zusätzlich die in Abbildung 4.17 dargestellten speziellen Arten von *Tasks* zur Modellierung verwendet werden. Hierbei spezifiziert der *SendMessageTask* einen *Task*, der eine Nach-

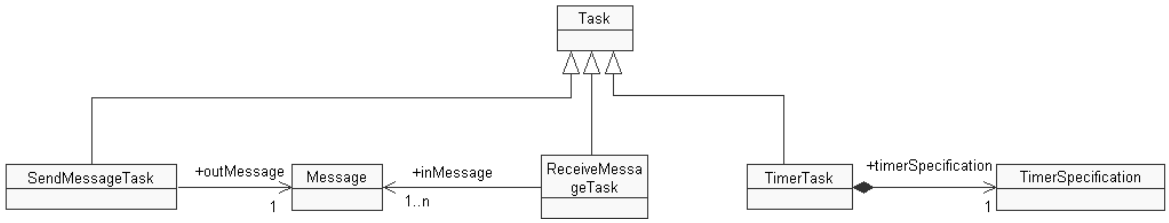

Abbildung 4.17: *Spezifikation von verschiedenen Task-Arten im konzeptuellen Metamodell.*

richt, deren Daten anhand eines Daten Input-Pin übergeben werden, über eine Prozess-Schnittstelle an ein Zielobjekt beispielsweise einen anderen Prozess versendet und nach erfolgreichen versenden, ohne auf eine Antwortnachricht zu warten, sofort beendet wird. Demgegenüber stellt der *ReceiveMessageTask* einen *Task* dar, der sobald er aktiviert wird, auf den Empfang spezifischer Nachrichten wartet, deren transportiertes Objekt über jeweils einen *Output*-Pin (für jede Nachricht einen Pin) den nachfolgenden Tasks zur Verfügung gestellt wird. Darüberhinaus existiert ein *TimerTask*, der auf den Eintritt eines spezifischen Zeitereignisses wartet, welches anhand einer *TimerSpecification* genau spezifiziert ist. Die Notation der jeweiligen Tasks ist in Abbildung 4.18 dargestellt.

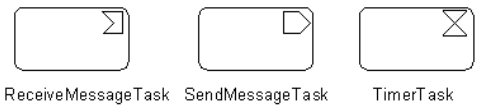

Abbildung 4.18: *Notation von SendMessage-, ReceiveMessage- und TimerTask.*

Im BPDM werden *SendMessageTasks* durch `SendObjectAction`s realisiert, die über ein `InputPin` alle notwendigen Informationen erhalten und diese in Form eines Request an das spezifizierte Zielobjekt versenden. Der *AcceptMessageTask* wird anhand der `AcceptEventAction` umgesetzt die nach ihrer Aktivierung auf den Eintritt eines spezifischen Ereignisses beispielsweise in Form des Aufrufs einer Prozess-Schnittstellenoperation wartet und deren Ausgabe über einen `OutputPin` emittiert wird.

ForEach Task

ForEach ist ein spezieller strukturierter Task, dessen Eingaben und Ausgaben durch einen Kollektionstyp bestimmt sind, der bisher im konzeptuellen Metamodell nicht weiter spezifiziert ist. Er führt für jedes in der Kollektion enthaltene Element, die durch ihn gekapselten *Steps* (ForEach-Block), aus, wobei jedes Ergebnis-Element in die Ausgabe-Kollektion an genau derselben Stelle, wo das zugehörige Eingabe-Element konsumiert wurde, emittiert wird. Das *expansionKind*-Attribut spezifiziert hierbei, ob entweder alle Ausführungen parallel, d.h. für jedes Element der Kollektion wird der ForEach-Block gestartet und alle Abarbeitungen laufen parallel oder seriell, d.h. für ein Element wird der ForEach-Block gestartet, sobald die Abarbeitung mit der Emittierung eines Ergebnis-Elements beendet ist wird für das nächste Element der Kollektion ein weiterer Block gestartet usw., ausgeführt werden. Hierbei ist der ForEach-Task beendet wenn die Abarbeitung des ForEach-Blocks für alle Elemente der Kollektion erfolgreich durchgeführt wurde.

Die Umsetzung dieses Konzepts im BPDM erfolgt anhand von `ExpansionRegion`s. Die Notation verdeutlicht die folgende Abbildung.

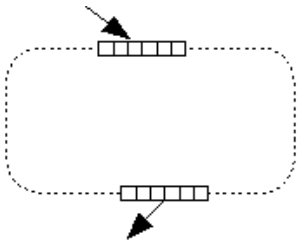

Abbildung 4.19: *Notation von ForEach-Task.*

Lebenszyklus von Steps

Steps werden gestartet, sobald über ein ungruppiertes *Input*-Pin bzw. einer Gruppe von *Input*-Pins über alle Pins ein Token eingeht bzw. vorhanden ist, dass während der Ausführung komplett von den *Steps* konsumiert wird.

Die Beendigung erfolgt, sobald sie ihre Aufgabe erfüllt haben, indem sie entweder über ein ungruppiertes *Output*-Pin oder über alle *Output*-Pins einer Gruppe Daten in Form von Token emittieren. Dies entspricht im wesentlichen der Aktivierungs-Semantik von Transitionen in Petri-Netzen.

Dieser Lebenszyklus ist in analoger Weise durch UML `Activities` definiert.

4.2.3 Kontrollfluss

Der Kontrollfluss beschreibt in welcher Reihenfolge die durch den Geschäftsprozess vorgesehenen Arbeitsschritte durchgeführt werden. Rein konzeptuell wird er aufgrund der Aktivierungssemantik von *Steps* durch Festlegung der von einem *Steps* besitzenden *Pins* sowie der die *Pin*s verbindenden Kanten spezifiziert. Abbildung 4.20 zeigt hierzu den Ausschnitt aus dem konzeptuellen Metamodell in dem die wesentlichen Elemente zur Prozessflussbeschreibung spezifiziert sind.

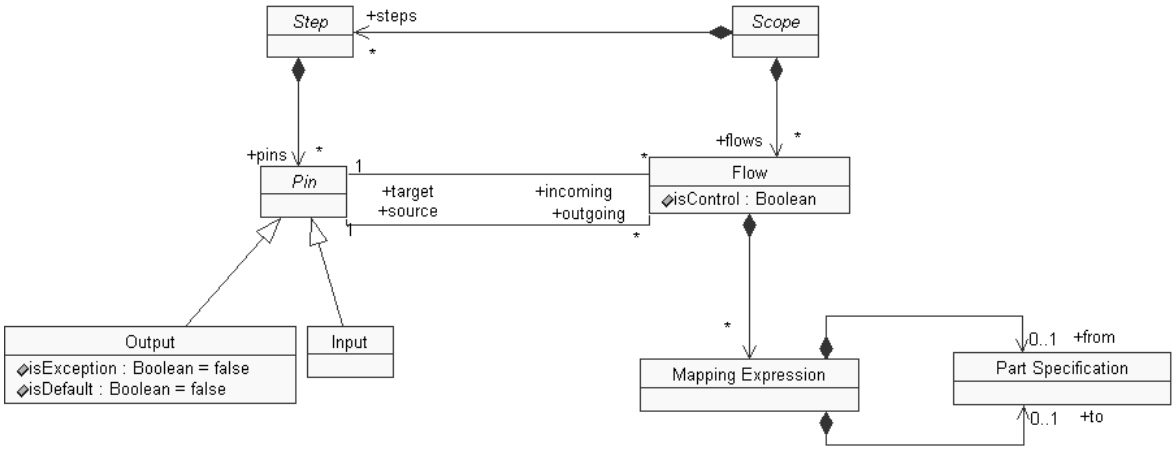

Abbildung 4.20: *Elemente zur Beschreibung des Prozessflusses.*

*Flow*s stellen gerichtete Kanten dar, die ein Quell- mit einem Ziel-*Pin* verbinden, über die entweder,

falls es sich um Kontrollflusskanten (*isControl=true*) handelt, Kontroll-Token oder bei Datenflusskanten (*isControl=false*) Daten-Token vom Quell- zum Ziel-*Pin* transportiert werden. Grundsätzlich verbinden Kontrollflusskanten lediglich Kontroll-Pins und Datenflusskanten ausschließlich Daten-Pins.

Im BPDM werden diese Konzepte mittels den in Aktivitätsdiagrammen vorhandenen Arten von `ActivityEdge`s realisiert. Hierbei werden Kontrollflusskanten anhand von `ControlFlow` und Datenflusskanten mittels `ObjectFlow` repräsentiert.

Da der Kontrollfluss im wesentlichen anhand der „Verpinnung“ von *Steps* spezifiziert wird, besteht aufgrund der unterschiedlichen Arten von *Pins* die Möglichkeit Kontroll- und Datenfluss sauber voneinander getrennt zu spezifizieren. Genauso gut kann der Kontrollfluss auch mit Hilfe des Datenflusses über die ausschließliche Verwendung von Daten-*Pins* und Datenflusskanten spezifiziert werden.

Im Folgenden wird die konzeptuelle Realisierung der klassischen strukturierten Kontrollflussstrukturen wie Sequenz, Alternative, Nebenläufigkeit und Schleifen anhand der Verwendung von Kontrollflusskanten und Kontroll-Pins dargestellt.

Sequenz

Eine sequentielle Abarbeitung von *Steps* wird erreicht, indem jeder *Step* über genau ein *Input-* als auch *Output-Pin* verfügt, wobei jeweils *Output-* und *Input*-Pin aufeinanderfolgender Steps über eine Kontrollflusskante miteinander verbunden werden. Die nachfolgende Abbildung zeigt schematisch die Modellierung einer Sequenz von *Tasks*.

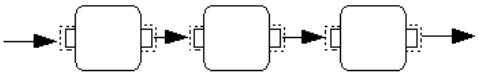

Abbildung 4.21: *Kontrollflussstruktur Sequenz.*

Verzweigungen

Verzweigungen werden aufgrund der impliziten ODER Semantik von *Pins* realisiert. Der Verzweigungsknoten (Decision-Knoten) verfügt über mehrere ungruppierte *Output*-Pins, die jeweils über Kanten mit den alternativen *Steps* bzw. Pfaden verbunden sind. Die Auswahlbedingungen für die einzelnen Pfade werden als Kanten-Bedingungen spezifiziert, die von den Eingabedaten des *Steps* und von *Variablen* abhängig sein können.

Der Verschmelzungsknoten (Merge-Knoten), bei dem alle alternativen Pfade zusammengeführt werden, wird mittels eines *Steps* realisiert, der über die gleiche Anzahl ungruppierter *Input*-Pins verfügt, wie der Verzweigungsknoten *Output*-Pins besitzt. Anhand der nachfolgenden Abbildung ist die Spezifikation von Verzweigungen veranschaulicht, cond1 und cond2 beschreiben hierbei die Auswahlbedingungen.

Nebenläufigkeit

Um mehrere Pfade im Kontrollfluss zu parallelisieren bedarf es eines *Steps* (Fork-Knoten), der mehrere gruppierte *Output*-Pins besitzt. Da bei Aktivierung über alle *Pins* der Gruppe Token emittiert

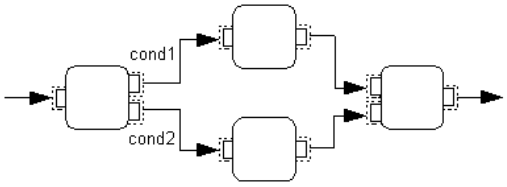

Abbildung 4.22: *Kontrollflussstruktur Verzweigung.*

werden, werden die nachfolgenden *Steps* der Pfade parallelisiert ausgeführt.

Um mehrere parallelisierte Pfade zu synchronisieren, ist in analoger Weise zum Verschmelzungknoten bei Verzweigungen ein *Step* (Join-Knoten) zu spezifizieren, der über genau die gleiche Anzahl an *Input*-Pins verfügt wie der Fork-Knoten, die in einer Gruppe zusammengefaßt sind.

Die Synchronisation zwischen einzelnen parallelen Pfaden wird mittels zusätzlichen Kontroll-Pins und Kontrollflusskanten realisiert. Die nachfolgende Abbildung zeigt die Modellierung von Nebenläufigkeit, zusätzlich wird *Task* B mit *Task* A synchronisiert.

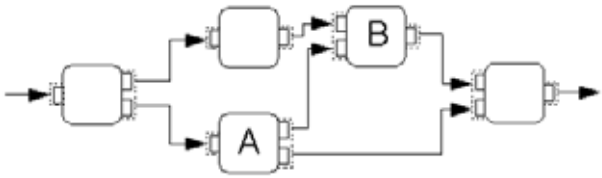

Abbildung 4.23: *Kontrollflussstruktur Nebenläufigkeit mit Synchronisation.*

Schleifen

Schleifen entsprechen im wesentlichen Verzweigungen, bei der eine Kante zu einem im Kontrollfluss früher liegenden *Step* „zurückgeführt“ wird. Die Schleifenbedingung entspricht der Verzweigungsbedingung. Abbildung 4.24 zeigt die Realisierung einer klassischen Repeat-Unitl Schleife.

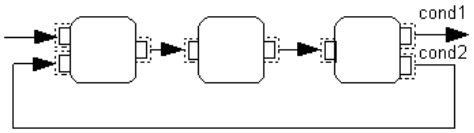

Abbildung 4.24: *Kontrollflussstruktur Schleife.*

Prinzipiell lassen sich mit den hier vorgestellten Sprachelementen weitaus komplexere Kontrollflussstrukturen, als die hier dargestellen, realisieren, da beispielsweise nicht alle Pfade einer Verzweigung in einem *Step* zusammengeführt werden müssen.

Gegenüber den bisher vorgestellten Kontrollflussstrukturen existieren zusätzliche Knoten, die speziell zur Beschreibung von Start- und Terminierungspunkten im Kontrollfluss von Geschäftsprozessen eingesetzt werden.

Start-, End- und FlowFinal-Knoten

Der *Start*-Knoten stellt einen Knoten dar, der einen zusätzlichen Startpunkt innerhalb des Prozesses beschreibt. Sobald der Prozess von außen aufgerufen (instanziert) wird, emittiert dieser über alle ausgehenden Kontrollflusskanten ein Kontrollfluss-Token an die nachfolgend auszuführenden *Steps*. Hierbei können grundsätzlich mehrere *Start*-Knoten im Prozess vorhanden sein, wodurch dieser von mehreren Punkten parallel gestartet wird.

Im Gegensatz zum *Start*-Knoten beendet der *End*-Knoten alle innerhalb des Prozesses ablaufenden Tokenübertragungen, wodurch der Prozess komplett terminiert wird. Eine lokale Termierung von Tokenübertragungen kann mit Hilfe des *FlowFinal*-Knoten realisiert werden, bei dem jeweils alle ihn über eingehende Kanten erreichenden Token „zerstört“ werden. *End*- als auch *FlowFinal*-Knoten verfügen lediglich über eingehende Kanten, wohingegen der *Start*-Knoten entweder eine oder mehrere ausgehende Kanten besitzt.

Im BPDM werden *Start*-Knoten durch `InitialNode`s, *End*-Knoten anhand von `ActivityFinalNode`s und *FlowFinal*-Knoten mittels `FlowFinalNode`s repräsentiert. Die nachfolgende Abbildung zeigt die Notation der jeweiligen Knoten.

Abbildung 4.25: *Notation von Start-, End- und FlowFinal-Knoten.*

4.2.4 Datenfluss und Datenbehandlung

Der Datenfluss beschreibt, welche Daten in Form von typisierten Token zwischen welchen *Steps* ausgetauscht werden. Hierzu werden wie bereits im Abschnitt über Kontrollflussmodellierung angedeutet, Datenflusskanten sowie Daten-Pins verwendet. Im Kontext der Spezifikation des Datenflusses existieren zusätzliche Modellelemente wie *Repository*s und *Variable*n, die im Prozessmodell zur Datenhaltung eingesetzt werden. Die folgende Abbildung zeigt einen Ausschnitt aus dem konzeptuellen Metamodell, in dem diese Elemente spezifiziert sind.

Daten-Pin und Item Definition

Daten-Pins stellen Eintritts- bzw. Austrittspunkte für *Steps* dar, über die spezifische Arten von Token emittiert bzw. konsumiert werden. Der Token-Typ, der aufgenommen bzw. abgegeben werden kann, wird anhand einer dem Pin zugeordneten *Item Definition* im konzeptuellen Metamodell spezifiziert. In der graphischen Notation wird dies durch die Zuordnung des Namens der *Item Definition* zum Daten-*Pin* dargestellt.

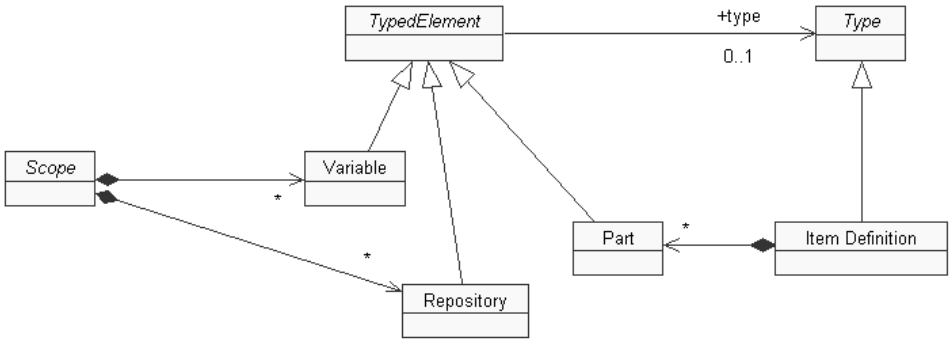

Abbildung 4.26: *Ausschnitt aus dem konzeptuellen Metamodell, in dem Repository, Variable, Item Definition und Part spezifiziert sind.*

Eine *Item Definition* kann ein Informationselement beschreiben z.B. ein Business Dokument oder ein Business Objekt. Sie besteht aus einer Menge von *Part*s, die jeweils die Einzelelemente des jeweiligen von *Steps* verarbeiteten Objekts charakterisieren. *Parts* können wiederum aus weiteren *Part*s bestehen. Beispielsweise könnte eine *Item Definition* eine Bestellung spezifizieren, die aus mehreren *Parts* beispielsweise einer Bestellnummer und einem Bestellnamen besteht.

Eine *Item Definition* wird im BPDM anhand einer mittels `<<entity>>` gestereotypten Klasse repräsentiert. Wie in diesem Zusammenhang *Parts* im BPDM realisiert werden, ist bisher nicht weiter spezifiziert.

Datenflusskanten und Mapping Expressions

Datenflusskanten sind gerichtete Kanten, die jeweils ein Daten-*Input*- und ein Daten-*Output*-Pin, die typkompatibel sein müssen, miteinander verbinden, über die Daten in Form von Daten-Token übertragen werden.

Sie verfügen gegenüber Kontrollflusskanten über die Möglichkeit sog. *Mapping Expression*s zu spezifizieren, anhand derer festgelegt werden kann, wie die über einen typisierten Pin emittierten Token in Token für einen anderen typisierten Ziel-Pin abgebildet werden können, indem einzelne Quell-*Part*s Ziel-*Part*s zugeordnet werden. Die Festlegung der Quell- und Ziel-*Part*s verschiedener *Item Definition*en erfolgt hierbei anhand von *Part Specification*.

Im BPDM werden *Mapping Expression*s anhand dem `ObjectFlow` zugeordneten Transformations-`Behavior` realisiert. Die Realisierung von *Part Specification* ist bisher nicht spezifiziert.

Die folgende Abbildung zeigt ein Beispiel der Spezifikation einer *Mapping Expression*. Hierbei emittiert ein Task eine Bestellung an einen Task, der jeweils nur eine Rechnung als Eingabe akzeptiert. Aufgrund der über die gestrichelte Linie mit der Datenkante verbundene *Mapping Expression* werden die einzelnen *Parts* der Bestellung auf äquivalente *Parts* einer Rechnung abgebildet, wodurch trotz Typinkompatiblität ein solcher Datenfluss realisierbar wird.

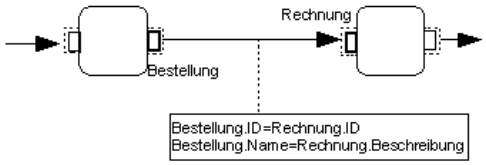

Abbildung 4.27: *Beispiel der Definition einer Mapping Expression.*

Repository

*Repository*s stellen im konzeptuellen Metamodell Elemente dar, die als Prozess übergreifende permante Datenspeicher für Token gleichen Typs, der anhand einer *Item Definition* spezifiziert ist, dienen und auf die innerhalb des Prozesses referenziert werden kann. Sie können eine gewisse vordefinierte Anzahl (Kapazität) von Token aufnehmen und sind mit *Task*s über Datenkanten verbunden. Der Zugriff auf *Repository*s kann entweder mittels Copy- oder Remove-Semantik erfolgen. Bei der Copy-Semantik werden die enthaltenen Token innerhalb des *Repository*s belassen und nur eine Kopie an den Empfängertask emittiert, wodurch andere *Task*s und Prozesse ebenfalls auf diese zugreifen können. Demgegenüber werden bei der Remove-Semantik emittierte Token aus dem *Repository* gelöscht. Ist ein Token bereits im *Repository* enthalten, so wird es überschrieben. Die folgende Abbildung zeigt eine mögliche Darstellung von *Repository*s. Die Bezeichnung entspricht dem Namen einer *Item Definition*, die den Typ des *Repository*s festlegt.

Die von Repositories aufgenommenen Token werden in einer sortierten Liste gehalten, bei der Token entweder am Anfang oder am Ende eingefügt werden. Es wird immer das Token emittiert das an erster Stelle der Liste steht, so dass abhängig der Einbringstrategie das Verhalten eines Stacks bzw. einer Warteschlange realisiert werden kann.

Abbildung 4.28: *Notation von Repositories.*

Bisher ist noch nicht spezifiziert, wie dieses Konzept im BPDM realisiert werden soll. Denkbar wären `CentralBufferNode`s oder `Property`s mit `AggregationKind=none`.

Variablen

*Variable*n werden verwendet um Prozess-spezifische Daten während der Ausführung zu speichern, anhand derer der Prozess-Zustand repräsentiert wird. Sie können als Prozess-Variablen, gültig für die gesamte Ausführungsdauer des Prozesses, oder für spezifisch strukturierte Tasks definiert werden. Hierbei gilt, dass nur *Task*s auf *Variable*n zugreifen können, die sich im selben Gültigkeitsbereich befinden. Bisher ist im konzeptuellen Metamodell der Zugriff von *Task*s auf *Variable*n nicht spezifiziert.

Das Konzept der *Variable* wird im BPDM bisher nicht weiter spezifiziert. Denkbar wären `Variable` oder `Property` mit `AggregationKind=composite`, für die bisher jedoch keine Notationen existieren.

4.2.5 Fehlerbehandlung

Zur Fehlerbehandlung wird das Konzept der Kompensation und Exceptions im BPDM unterstützt, wobei zur Zeit lediglich Exceptions vom konzeptuellen Metamodell genauer spezifiziert werden. Die Realisierung des Konzepts der Kompensation ist bisher lediglich textuell in [16] beschrieben.

Exceptions

Exceptions werden genutzt um Ausnahmen, Fehler (Exceptions), die bei der Ausführung eines *Steps* entstehen durch die Definition eines Fehlerbehandlungs-*Steps* abzufangen und entsprechend zu behandeln. Hierzu werden sog. Exceptions *Output*-Pins, *Output*-Pins mit *isException=true* spezifiziert, die im Fehlerfall ein Token, welches den genauen Fehler näher charakterisiert, emittiert. Über die Kopplung des *Output*- mit einem *Input*-Pins eines anderen *Steps* kann dieser als ExceptionHandler fungieren. Die Notation von Exception *Output*-Pins ist in Abbildung 4.29 dargestellt. Hierbei stellt Task B den ExceptionHandler dar, der vom Task A, falls dieser einen Fehler anhand eines über das durch ein Dreieck markierte Exception *Output*-Pin emittierte Token signalisiert, aufgerufen wird.

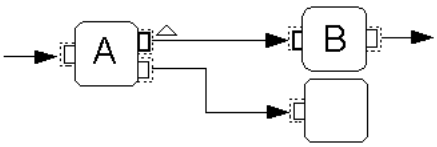

Abbildung 4.29: *Beispiel der Spezifikation einer Exception sowie eines Behandlungstasks.*

Die Umsetzung dieses Konzepts erfolgt im BPDM über `OutputPins`, die mittels `isException=true` als Exception `OutputPins` spezifiziert werden.

Kompensation

Ein möglicher Ansatz zur Realisierung des Kompensations-Konzepts im BPDM ist, *Steps* (primären *Steps*) sog. Kompensations-*Steps* zuzuordnen, die für den jeweiligen primären *Step* installiert werden, sobald dieser erfolgreich beendet wird. Hierbei werden primäre mit Kompensations-*Steps* anhand von Kontollflusskanten miteinander gekoppelt. Wird anhand eines über ein Exception *Output*-Pin emittierten Token ein Fehler in der Abarbeitung des die zu kompensierenden *Steps* enthaltenen strukturierten Tasks signalisiert, werden alle Kompensations-*Steps* in der umgekehrten Installationsreihenfolge ausgeführt. Zusätzlich werden alle eingebetteten strukturierten Tasks kompensiert. Die folgende Abbildung ziegt hierzu ein Beispiel.

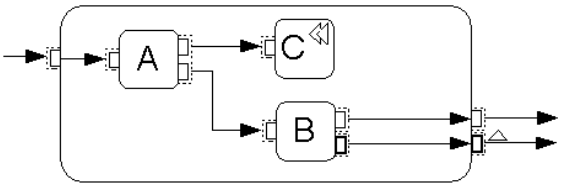

Abbildung 4.30: *Einfaches Beispiel der Verwendung von Kompensations-Steps.*

Wird beispielsweise *Task* A erfolgreich ausgeführt, emittiert dieser zwei Kontrolltoken. Ein Kontrolltoken führt zur Aktivierung von *Task* B und eins zur Aktivierung von *Task C*. Da es sich bei *Task*

C um einen Kompensations-*Step* handelt, wird dieser nicht ausgeführt, sondern als Kompensations-*Step* für *Task* A installiert. Würde der strukturierte Task anhand eines Fehlers, verursacht durch *Task* B, abgebrochen, würde automatisch der für *Task* B installierte Kompensations-*Step Task* C aktiviert, der den durch *Task* A gemachten Arbeitsschritt kompensiert. Im konzeptuellen Metamodell sind Kompensations-Steps bisher nicht berücksichtigt.

4.2.6 Kommunikationsmodell

Wie bereits in einem der vorherigen Abschnitte erwähnt, wird zur Kommunikation zwischen Prozessen und Partnern ein auf den Austausch von Nachrichten basiertes konzeptuelles Kommunikationsmodell verwendet, mit dem Prozesse und Partner lose miteinander gekoppelt werden.

Im wesentlichen verläuft eine Kommunikation nach folgenden Muster ab: Ein Task, speziell ein *SendMessageTask* sendet eine Nachricht an einen spezifischen Partner um einen Dienst in Anspruch zu nehmen. Der Partner empfängt die Nachricht und reagiert mittels eines *ReceiveMessageTask*s und der Durchführung nachfolgender *Steps* mit beispielsweise einem abschließenden *SendMessageTask*.

Dieses rein konzeptuelle Kommunikationsmodell wird im BPDM wie folgt umgesetzt. Will Beispielsweise eine `Action` ein Verhalten in Form einer Operation eines entfernten oder lokalen Objekts aufrufen, wird nach Aktivierung der `Action` im Aktivitätsdiagramm ein spezifisches Event erzeugt, anhand dessen dies signalisiert wird (für die `SendObjectAction` ist dies ein `SendInvocationEvent`). Dieses Event führt zur Erzeugung eines spezifischen Request-Objekts auf Senderseite, das alle notwendigen Informationen über Absender, Empfänger und Daten enthält. Wird dieser Request durch das Empfängerobjekt (Prozess-Klasse o. bel. anderer Partner) empfangen, so wird ein zugehöriges Ereignisobjekt , welches in Form eines `Triggers` repräsentiert wird, erzeugt und in einen Ereignispool abgelegt, auf das entweder anhand eines beinhaltender `AcceptEventAction` oder einer zugeordneten Operationen mit einem spezifische Verhalten reagiert werden kann.

4.2.7 Prozess Lebenszyklus

Die Ausführung eines spezifizierten Prozesses wird gestartet, sobald ein *ReceiveMessageTask* , der keine eingehenden Kanten besitzt, das Ziel einer eingehenden Nachricht (initiale Request Nachricht) ist, die nicht mit einer bereits in der Ausführung befindlichen Instanz dieses Prozesses korreliert ist, indem eine neue Instanz des Prozesses erzeugt und mit der Abarbeitung der einzelnen *Steps* begonnen wird. Demgegenüber wird eine sich in der Ausführung befindliche Prozessinstanz beendet, sobald entweder ein *End*-Knoten erreicht wird oder alle Token Flüsse und damit die Ausführung aller *Steps* beendet sind.

Aufgrund der möglichen Instanzierung von Prozessen an unterschiedlichen Lokationen sowie dem gleichzeitigen Ablaufen mehrerer Instanzen ein und desselben Prozesses müssen zusätzliche Informationen spezifiziert werden, um die Partner eines Prozesses lokalisieren zu können sowie zu entscheiden, welche Nachricht an welche Instanz eines Prozesses weiterzuleiten ist.

Partner Lokation

Bisher ist im konzeptuellen Metamodell nicht weiter spezifiziert, wie Partner Lokationsinformationen beschrieben werden können. Im Allgemeinen sollen aber die nachfolgenden vier Möglichkeiten für die Modellierung unterstützt werden:

- die Lokationsinformationen brauchen nicht modelliert werden, die Umgebung, in der der Prozess abläuft, wird „irgendwie wissen“l, wo der jeweilige Partner lokalisiert ist.

- sie werden im Modell erfasst, beispielsweise anhand der Spezifikation von Attributen, die die Lokalisierung von Partnern beschreiben.
- sie werden mittels ein Selektions-Algorithmus spezifiziert, der anhand einer Menge von Attributen und auszutauschenden Nachrichten, aufgrund der Kollaborationsdefinition, den Partner identifiziert.
- oder es wird ein Publish-Subscribe Mechanismus verwendet.

Partner Korrelation

Da wie bereits in BPEL4WS einzelne Prozessinstanzen im BPDM nicht direkt-adressierbar sind, müssen zwischen Instanzen auszutauschende Nachrichten anhand der Nachrichten selbst den Instanzen zugeordnet werden, was als Korrelation bezeichnet wird. Hierbei wird vom BPDM zwischen impliziter und expliziter Korrelation unterschieden, die im Folgenden näher charakterisiert werden.

Implizite Korrelation meint, dass miteinander kollaborierende Prozessinstanzen anhand initialer Request Nachrichten für die Dauer der Kollaboration automatisch miteinander gekoppelt werden. D.h. zwei Prozesse interagieren anhand einer spezifizierten Kollaboration. Sobald einer anhand eines Request die Kollaboration initiiert, werden beide für die Dauer der Kollaboration miteinander gekoppelt, so dass weitere Nachrichten im Kontext des Kollaborationsprotokolls an die jeweilige gebundene Prozessinstanz weitergeleitet werden.

Im Gegensatz dazu besteht auf konzeptueller Ebene die Möglichkeit den vom Prozess empfangbaren Nachrichten Korrelationprädikate (*correlationPredicate*, siehe Abbildung 4.14) zuzuordnen, anhand derer im Modell explizit festgelegt werden kann, an welche Prozessinstanz eine empfangene Nachricht weitergeleitet werden soll (explizite Korrelation). Es handelt sich hierbei um Ausdrücke, die aus elementaren Bedingungen, die auf Prozess-*Variablen*, Konstanten und Nachrichtenteile der ihnen zugeordneten Nachricht referenzieren, bestehen, die mittels boolschen Operatoren ($\wedge, \vee$) miteinander verknüpft werden und für eine konkrete Nachricht und Prozessinstanz entweder zu wahr oder falsch evaluiert werden. Ein Korrelationsprädikat ist beispielsweise

$$bestellID = Bestellung.ID \wedge auftraggeber = Bestellung.Absender$$

,wobei $bestellID$ und $auftraggeber$ interne Prozess-*Variable*n darstellen, $Bestellung$ die Nachricht bezüglich der das Korrelationsprädikat definiert ist sowie $Bestellung.ID$ und $Bestellung.Absender$ die *Part*s der Nachricht auf die Bezug genommen wird. Eine mögliche graphische Notation von Korrelationsprädikaten zeigt die folgende Abbildung anhand des dargestellten Beispiels.

Abbildung 4.31: *Mögliche graphische Notation von Korrelationsprädikaten.*

Wird das Korrelationsprädikat für eine empfangene Nachricht für genau eine Prozessinstanz zu wahr evaluiert, wird die Nachricht an diese Prozessinstanz weitergeleitet. Für den Fall, dass für mehrere Prozessinstanzen oder für keine das Prädikat zu wahr evaluiert wird, kann durch Angabe zusätzlicher

Attribut	**Wert**	**Semantik**
no correlation matches berücksichtigt, wenn das Korrelationsprädikat für keine Prozessinstanz zu wahr evaluiert wird.	create new instance	Erzeugt eine neue Prozessinstanz.
	raise exception	Eine „no correlation matches“ Exception wird erzeugt.
	ignore	Die Nachricht wird ignoriert.
multiple correlation matches berücksichtigt, falls für mehrere Instanzen das Korrelationsprädikat zu wahr evaluiert wird.	deliver to all	Die Nachricht wird an alle Prozessinstanzen weitergeleitet, die zu wahr evaluiert wurden.
	delivery to any	Die Nachricht wird an irgendeine Prozessinstanz weitergeleitet, die zu wahr evaluiert wurde.
	raise exception	Eine „multiple correlation matches“ Exception wird erzeugt.
	ignore	Die Nachricht wird ignoriert.

Tabelle 4.2: *Attribute für Korrelationsprädikate, ihre möglichen Werte sowie deren Semantik.*

Attribute des Korrelationsprädikats das jeweilige Verhalten der Ausführungsumgebung bestimmt werden. Tabelle 4.2 zeigt die beiden Attribute, die verschiedenen von ihnen annehmbaren Werte sowie deren Semantik.

Bisher ist die Realisierung von Korrelationsprädikaten im BPDM nicht spezifiziert.

4.2.8 Ressourcen

Wie in einem der vorherigen Abschnitte erwähnt, stellen *Task*s die Elemente zur Beschreibung der von einem Geschäftsprozess durchzuführenden atomaren Arbeitsschritte, dar die lediglich Aktionen bzw. Tätigkeiten wie beispielsweise „Prüfe Bestellung“ spezifizieren. Da jede Ausführung eines Arbeitsschritts gewisser Ressourcen bedarf, beispielsweise einer Person, die die Aktivität ausführt, müssen diese zusätzlich im Modell spezifiziert werden. Hierzu werden jedem Task sog. *ResourceRequirement*s zugeordnet, die entweder direkt Ressourcen spezifizieren, die der *Task* zur Realisierung benötigt oder Rollen, die die Anforderungen zur Realisierung eines *Tasks* beschreiben. Um die unterschiedlichen Arten von *RescourceRequirements* zu verstehen, werden im Folgenden die im konzeptuellen Metamodell spezifizierten Ressource-Arten (Abbildung 4.32) näher dargestellt.

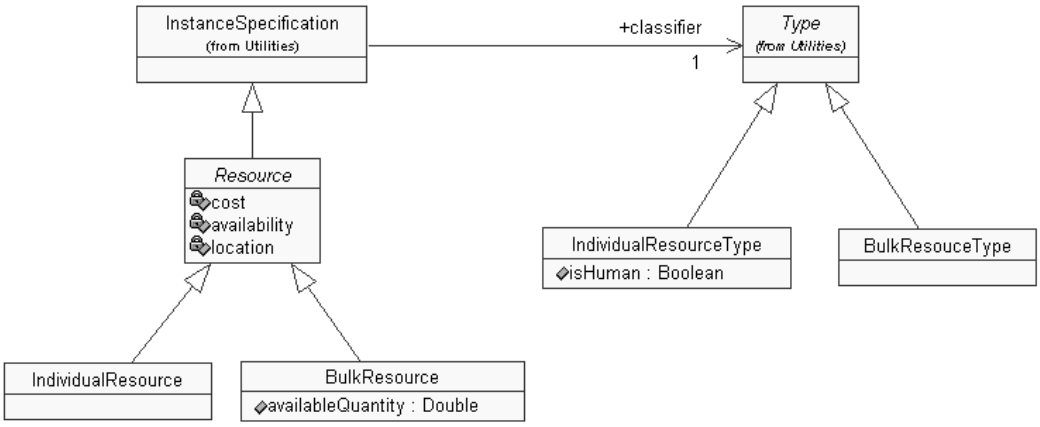

Abbildung 4.32: *Elemente zur Ressourcen-Spezifikation im konzeptuellen Metamodell.*

Grundsätzlich werden zwei unterschiedliche Arten an Ressourcen unterschieden: Individuelle- (*IndividualResource*) und Bulk-Ressourcen (*BulkResource*). Individuelle Ressourcen sind dadurch gekennzeichnet, dass sie eine Identität besitzen und entweder menschlicher (*isHuman=true*) oder nichtmenschlicher (*isHuman=false*) Natur sind, wohingegen Bulk-Ressourcen über keine Identität verfügen und in gewissen Mengen benötigt werden. Beispielsweise handelt es sich bei dem Arbeiter Hans Müller um eine individuelle Ressource, wohingegen 100l Öl einer Bulk-Ressource entsprechen. Jede im Modell spezifizierte Ressource ist zusätzlich durch einen Ressource-Typ charakterisiert, der in diesem Sinne als UML `Classifier` verstanden werden kann, wodurch wiederum zwischen Individuellen- (*IndividualResourceType)* und Bulk-Typen (*BulkResourceType*) unterschieden wird.

Eine Möglichkeit die vom Task benötigten Ressourcen-Anforderungen zu spezifizieren, liegt darin mittels *BulkResourceRequirements* oder *IndividualResourceRequirements* (Abbildung 4.33) entweder direkt Ressourcen oder Ressourcen-Typen (*BulkResourceType*, *IndividualResourceType*) zu spezifizieren. Beispielsweise könnte eine individuelle Ressourcen-Anforderungen, spezifiziert bezüglich eines Ressourcen-Typs, sein: „Ein Arbeiter dessen Vorname Hans ist“ wodurch der Task, der diese Ressourcen-Anforderung besitzt, lediglich durch Arbeiter, die über den Vornamen Hans verfügen, ausgeführt werden kann.

Eine andere Möglichkeit der Spezifikation von Ressourcen-Anforderungen besteht darin, nicht Ressourcen direkt oder anhand eines Typs zu spezifizieren, sondern lediglich Eigenschaften sog. Rollen (*RequiredRole*), die von *Task*s benötigt und durch Ressourcen zur Verfügung gestellt werden. Dadurch

werden die von einem *Task* benötigten Ressourcen nicht auf einen Ressourcen-Typ festgelegt, sondern es können Ressourcen unterschiedlicher Typen, die die Rolle erfüllen beispielsweise für die Ausführung des Tasks in Frage kommen.

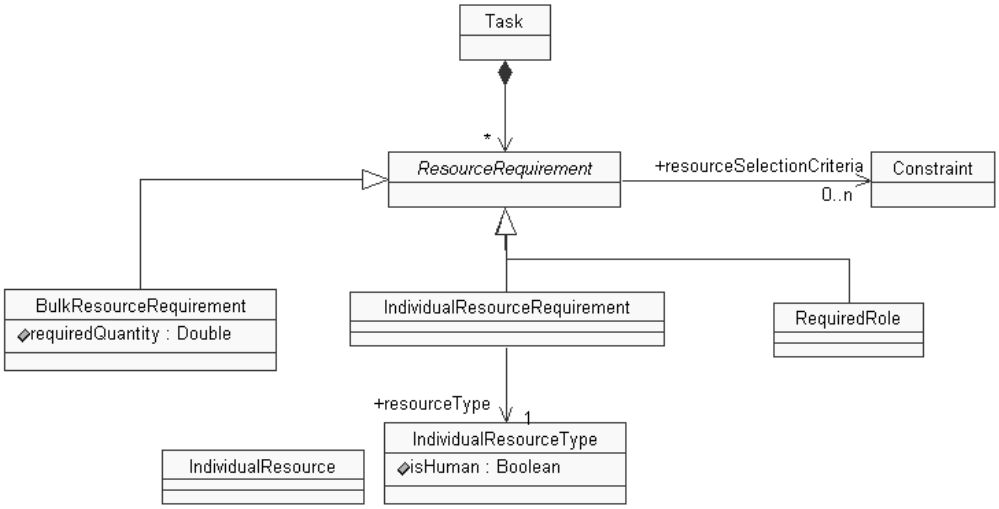

Abbildung 4.33: *Ausschnitt aus dem konzeptuellen Metamodell, in dem die Elemente zur Beschreibung von Ressourcen-Anforderungen spezifiziert sind.*

Die Umsetzung dieser Konzepte und deren graphische Notation sind bisher nicht weiter spezifiziert.

4.3 BPDM, BPEL4WS und MDA

Die MDA beschränkt sich nicht nur auf den Bereich der Softwareentwicklung, sondern kann speziell als Methode für eine konsistente Entwicklung und Implementierung von Geschäftsprozessen angesehen werden, bei der Geschäftsprozesse unabhängig einer Ausführungsplattform als CIM- und PIM-Modelle spezifiziert werden, die anschließend in PSMs, aus denen die ausführbaren IT Artifakte für die Implementierung abgeleitet werden, transformiert werden. Dies hat den Vorteil, dass der Graben, der zwischen Business Analysten und IT-Entwicklern bezüglich der Entwicklung und Implementierung von Geschäftsprozessen existiert, durch eine stufenweise Verfeinerung der Geschäftsprozessmodelle in Richtung einer konkreten Implementationstechnologie zunehmend verringert wird.

Um den MDA-Ansatz auf den Bereich der Entwicklung und Implementierung von Geschäftsprozessen zu übertragen, bedarf es geeigneter Sprachen bzw. Metamodelle, mit denen sich jeweils auf CIM-, PIM- und PSM-Ebene Geschäftsprozesse spezifizieren lassen. Für PIM-Ebene bietet sich hierfür das BPDM an, da es sich um ein UML Profile handelt, welches direkt für die PIM-Ebene entwurfen wurde und mit dem Geschäftsprozesse unabhängig einer bestimmten Ausführungsplattform bzw. Implementationstechnologie spezifiziert werden können. Für die PSM-Ebene können mittlerweile zahlreiche Metamodelle zur Spezifikation von Geschäftsprozessen bezüglich einer konkreten Implementierungstechnologie verwendet werden, beispielsweise ein UML Profile für EJBs oder ein Metamodell für BPEL4WS. BPEL4WS bietet sich hierbei besonders an, da es mittlerweile eine große Bedeutung als Sprache zur Beschreibung ausführbarer Geschäftsprozesse auf Basis von Web Services erlangt hat, für

die mittlerweile ein UML Profile [32] sowie im Umfeld von Eclipse ein EMF Modell[2] existiert, mittels denen sich BPEL4WS Prozesse spezifizieren lassen. Die folgende Abbildung zeigt eine schematische Darstellung der im Kontext der MDA vorhandenen Modellarten sowie der Einordnung von BPDM und BPEL4WS.

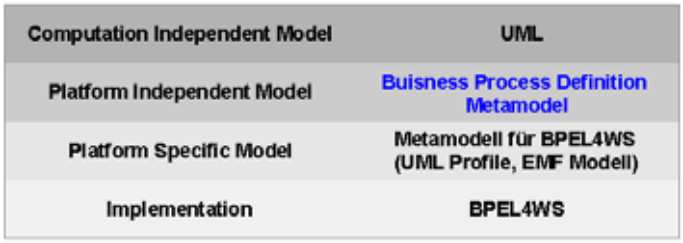

Abbildung 4.34: *Einordnung des BPDM und BPEL4WS in die MDA.*

Da das zentrale Systemkonstruktionsparadigma der MDA die Abbildung von abstrakteren Modellen in näher an einer Implementierungstechnologie liegende Modelle ist, ist speziell im Fall der Implementierung von Geschäftsprozessen mittels MDA für die eingesetzten Metamodelle, speziell BPDM und BPEL4WS, zu spezifizieren, wie diese ineinander abgebildet werden können und an welchen Stellen der Abbildung Limitationen und Komplikationen vorhanden sind, auf die im nächsten Kapitel näher eingegangen wird.

[2]Eclipse Modeling Framework, Java Implementierung der wesentlichen Elemente der MOF API, basierend auf der Open-Sourcen Plattform Eclipse, mit dem sich in Form eines EMF Modells Metamodelle spezifizieren lassen, die beispielsweise im Kontext der MDA verwendbar sind.

Kapitel 5

Abbildung von BPDM-Prozessen in BPEL4WS

Im Kontext des MDA Ansatzes ist es notwendig Modelle, die auf PIM Ebene definiert wurden, für die spätere Implementierung auf eine spezifische Ausführungplattform, d.h. in ein PSM, abzubilden. Für Geschäftsprozessmodelle spezifiziert anhand des BPDM kann dies speziell BPEL4WS bzw. ein dafür konzipiertes Metamodell sein. Daher ist es notwendig aufzuzeigen, in welchen Bereichen der Abbildung von BPDM Prozessen in BPEL4WS grundsätzlich Limitationen und Komplikationen vorhanden sind und wie diese mit Bezug auf die Ausrichtung der Zielsprache gelöst werden können[1].

Dieses Kapitel beschreibt die verschiedenen Komplikationen und Limitationen bei der Abbildung von BPDM- in BPEL4WS-Prozessen und orientiert sich hierbei an dem i Kapitel 4 beschriebenen konzeptuellen Metamodell des BPDM sowie an der in Kapitel 3 charakterisierten BPEL4WS Spezifikation 1.1. Im ersten Abschnitt wird hierbei die Abbildung von BPDM in BPEL4WS anhand des konzeptuellen Metamodells grob charakterisiert. Der zweite Abschnitt bietet eine detaillierte Darstellung, der bei der Abbildung vorhandenen Limitationen und Komplikationen.

5.1 Charakterisierung der Abbildung von BPDM in BPEL4WS

Dieser Abschnitt bietet einen Überblick über die Abbildung der zur Geschäftsprozessmodellierung im BPDM zur Verfügung stehenden Konzepte in BPEL4WS, indem anhand der Elemente des konzeptuellen Metamodells skizziert wird, wie diese mittels BPEL4WS semantisch äquivalent realisiert werden können. Damit handelt es sich, im Kontext der MDA, um die Spezifikation einer Metamodell Transformation, die aufgrund des bisherigen Status des konzeptuellen Metamodells in diesem Abschnitt noch unvollständig charakterisiert ist.

Im ersten Teil erfolgt die Darstellung der Umsetzung der wesentlichen Konzepte zur Beschreibung der externen Struktur und des externen Verhaltens von BPDM Prozessen in BPEL4WS, anschließend die Charakterisierung der Abbildung der für die Datenbehandlung wesentlichen Elemente sowie die Darstellung der Realisierung der zur Beschreibung des Prozessverhaltens grundlegenden Konstrukte. Abgeschlossen wird dieser Abschnitt mit der Charakterisierung der Abbildung der für das globale Prozessverhalten möglichen Kontrollflussstrukturen.

[1] Die Betrachtung der reinen Sprachdefinition von BPEL4WS ist äquivalent zur Untersuchung anhand eines für BPEL4WS konzipierten Metamodells, welches die Konzepte und Elemente sowie die Struktur von BPEL4WS Prozessen syntaktisch beschreibt.

Zur übersichtlicheren Beschreibung der Abbildung erfolgt die Darstellung in tabellarischer Form.

5.1.1 Prozessdefinition

Element des konzeptuellen Metamodells	**XSD-, WSDL- oder BPEL4WS-Konstrukt**
Package	Bisher ist im konzeptuellen Metamodell kein äquivalentes Element definiert, mit dem sich Modelle organisieren und damit übersichtlicher gestalten lassen. Aufgrund der Umsetzung des konzeptuellen Metamodells in UML kann aber davon ausgegangen werden, dass das in UML hierzu typische Package Konstrukt verwendet wird. Diese werden als Datei-Verzeichnisse abgebildet, in denen die jeweiligen BPEL4WS-, WSDL- und XML-Schema Dateien, die aus der Abbildung der im Package definierten Prozessmodellelemente resultieren, abgelegt werden. Der Name des Package entspricht dem Verzeichnisnamen. Zusätzlich legt der vom Package definierte Namensraum den Prefix für alle notwendigen Namensraum-Definitionen fest.
Prozess (isAbstract=false)	Wird als ausführbarer BPEL4WS Prozess abgebildet. Der Name des Prozesses entspricht dem Namen des BPDM *Prozess*es.
Prozess (isAbstract=true)	Wird als abstrakter BPEL4WS Prozess abgebildet, der Name des Prozesses entspricht dem Namen des BPDM *Prozess*es. Aufgrund von Beschränkungen in BPEL4WS ergeben sich hinsichtlich der Abbildung abstrakter BPDM Prozesse Schwierigkeiten. Siehe hierzu Abschnitt 5.2.1.
realizes-Assoziation	In BPEL4WS kein äquivalentes Element vorhanden. Siehe hierzu Abschnitt 5.2.1.
kollaborativer Prozess	Wird als solches nicht von BPEL4WS unterstützt. Siehe hierzu Abschnitt 5.2.2.
Interface	Wird als WSDL-portType beschrieben, dessen Name dem *Interface*-Namen mit dem Suffix PortTyp entspricht.
Interface assoziierte *in-Message*	Wird als one-way WSDL-Operation des korrespondierenden WSDL-portType des die *Message* enthaltenen *Interface* abgebildet. Der *Message* Name entspricht dem Operationsnamen. Die Input WSDL-Message der Operation ist jeweils die aus Abbildung der zugehörigen *Item Definition* resultierende WSDL-Message.
correlationPredicate	Können nur sehr eingeschränkt abgebildet werden, da ihre Mächtigkeit die von Korrelationsmengen übertrifft. Siehe hierzu Abschnitt 5.2.4. Jede dem Task, der auf eine mit einem Korrelationsprädikat versehene Message reagiert , korrespondierende BPEL4WS-Aktivität wird an der dem Korrelationsprädikat korrespondierenden Korrelationmenge mit `initiate="no"` korreliert.
Connector	Wird als PartnerLink Type Definition abgebildet. Referenzierte PortTyps sind die korrespondierenden PortTyp Definitionen, der durch den *Connector* verbundene *Interfaces*, zusätzlich werden PartnerLink Definitionen auf Seiten beider Prozesse erzeugt, die jeweils eine der durch den *Connector* verbundenen Rollen inne haben.

Property, Subprozesse	Subprozesse werden von BPEL4WS nicht unterstützt. Siehe hierzu Abschnitt 5.2.3.

5.1.2 Datenbehandlung

Element des konzeptuellen Metamodells	**XSD-, WSDL- oder BPEL4WS-Konstrukt**
Item Definition	Wird als WSDL-Message repräsentiert. Der Name der WSDL-Message entspricht dem Namen der *Item Definition* mit beispielsweise dem zusätzlichen Suffix Msg.
Part	Wird als Part einer WSDL-Message realisiert. Name des Parts ist der Name des *Parts* im BPDM. Da Parts wiederum mehrere Parts als Member enthalten können, wird lediglich die erste Ebene in Form von WSDL-Message Parts abgebildet. Alle weiteren werden in Form von XML-Schemas realisiert.
Pin (Daten-Pin)	BPEL4WS Variable getypt mittels WSDL-Messages.
Pin (Kontroll-Pin)	Brauchen in BPEL4WS nicht durch ein geeignetes Konstrukt berücksichtigt werden, da sie den Kontrollfluss des Prozesses mit bestimmen und anhand der jeweiligen realisierten Kontrollflussstrukturen indirekt bei der Abbildung Berücksichtigung finden.
Output-Pin (isException=true)	Werden in BPEL4WS mittels `faultHandlers` realisiert, die im Kontext eines strukturierten Tasks dem jeweiligen korrespondierenden `scope` zugeordnet werden. Bei Tasks, die mittels `invoke`-Aktivität realisiert werden, werden sie anhand des `<catch faultName="...">..</catch>` Statements in BPEL4WS abgebildet.
Flow (*isControl=true*)	Wird durch BPEL4WS Kontrollflussstrukturen im Kontext der Beschreibung des globalen Prozessverhaltens mit berücksichtigt.
Flow (*isControl=false*)	`assign`-Aktivität, die die *Pins* korrespondierenden Variablen ineinander abbildet. Dies ist notwendig, da jedes *Pin* anhand einer Variable repräsentiert wird. Stimmen die Typen der Pins korrespondierenden Variablen überein, werden die Variablen zusammengefaßt, die als Aus- bzw. Eingabevariable verwendet wird. Zusätzlich wird der damit realisierte Kontrollfluss im Kontext der Beschreibung des globalen Prozessverhaltens mit berücksichtigt.
Respository	Keine äquivalente Entsprechung in BPEL4WS. Siehe hierzu Abschnitt 5.2.7.
Variable	Werden als BPEL4WS Variablen abgebildet. Der Name der Variablen entspricht dem Name der BPDM *Variable*. Der Typ der Variable wird anhand der, der *Variable* zugeordneten *Item-Definition* korrespondierenden WSDL-Message bzw. falls es sich um einen atomaren Typ handelt als einfacher XML Schema Typ, definiert.
Mapping Expression	`assign`-Aktivität mit einem `copy`-Statement.Die `from-spec` entspricht der Form `<from variable="name"part="...">`. Die `to-spec` ist analog. Die referenzierten Variablen sind die jeweils durch den Object-Flow verbundenen korrespondierenden Pin-Variablen.
Part Specification	Kann als XPath-Ausdruck, der jeweils das `part`-Attribut der `from-spec` und `to-spec` charakterisiert, abgebildet werden.

5.1.3 lokales Prozessverhalten

Tasks

Element des konzeptuellen Metamodells	**XSD-, WSDL- oder BPEL4WS-Konstrukt**
Task	Entweder als synchrone Request-Response `invoke`-Aktivität abbildbar, hierbei wird der Operationsname aus dem *Task*-Namen abgeleitet, darüber hinaus sind zusätzliche Informationen bei der Abbildung zu spezifizieren bzgl. PartnerLinkTyp, PartnerLink und PortType, oder als unspezifiziertes Muster, das bei der Abbildung genauer, durch beispielsweise einen Benutzer, zu spezifizieren ist, d.h. der Task als „Black Box“. Bei der Realisierung mittels `invoke`-Aktivität ergeben sich grundsätzlich Schwierigkeiten bezüglich der Behandlung mehrerer Daten Input- und Output-*Pins* (Datenfluss) sowie der Abbildung von manuellen Tasks mit *isHuman=true*. Siehe hierzu Abschnitt 5.2.6 und 5.2.9.
SendMessageTask	One-way `invoke`-Aktivität, deren Operationsname dem Namen des *SendMessageTask* entspricht. In Analogie zu *Task*s sind zusätzliche Informationen zu spezifizieren.
ReceiveMessageTask (reagiert auf eine Message)	`receive`-Aktivität. Der Operationsname entspricht dem Namen der *Message*, auf deren Empfang gewartet wird. In Analogie zu *Task*s sind zusätzliche Informationen zu spezifizieren.
ReceiveMessageTask (reagiert auf mehrere Messages)	`pick`-Aktivität, die mehrere `onMessages`-Statements besitzt, deren Operationsnamen den jeweilige *Message*-Namen entsprechen.
TimerTask	Wurde im konzeptuellen Metamodell bisher noch nicht weiter spezifiziert. Wird vorraussichtlich anhand der `wait`-Aktivität in BPEL4WS realisierbar sein.
TimerSpecification	Bestimmt den Deadline- oder Duration-Wert Ausdruck, der dem korrespondierenden *TimerTask* korrespondierenden `wait`-Aktivität.
Scope	Alle im konzeptuellen Metamodell von BPDM beschriebenen *Scopes* werden als entsprechende BPEL4WS `scope` abgebildet. Alle enthaltenen Elemente werden in ihrer Umsetzung in BPEL4WS in den korrespondierenden Scope eingebettet.
StructuredTask	`scope`-Aktivität. Alle durch einen *structuredTask* zusammengefaßten *Task*s werden als in den Scope eingebettete BPEL4WS Aktivitäten abgebildet. Der Name des `scope`s entspricht dem Name des *StructuredTask*.
ForEach-Task	Zur Zeit in BPEL4WS kein äquivalentes Konstrukt vorhanden. Schwierigkeiten bei der Abbildung des parallelisierten *ForEach-Task*. Siehe hierzu Abschnitt 6.2.

Prozessfluss-Elemente

Element des konzeptuellen Metamodells	**XSD-, WSDL- oder BPEL4WS-Konstrukt**
Start	In BPEL4WS existiert hierzu keine korrespondierende Aktivität. Wird von BPEL4WS aber implizit durch die erste auszuführende Basis-Aktivität ausgedrückt. Falls mehrere *Start*-Knoten existieren, werden die nachfolgenden auszuführenden *Tasks* mittels einer `flow`-Aktivität, die zu Beginn des Prozess gestartet wird, parallelisiert.
End	Wird als `terminate`-Aktivität repräsentiert.
FlowFinal	Wird abhängig seiner Position im BPDM Prozess wie folgt abgebildet: befindet es sich in einem nichtparallelisierten Pfad einer Verzweigung, wird es mittels `terminate`-Aktivität abgebildet. Ist es demgegenüber in einer Parallelisierung, wird der gesamte parallelisierte Pfad durch einen `scope` charakterisiert, der *FlowFinal*-Task als `throw`-Aktivität, die einen leeren Fault Handler des Scopes aufruft. Tritt keiner dieser Fälle auf, wird es irgnoriert.

5.1.4 globales Prozessverhalten

klassische Kontrollflussstrukturen

Kontrollflussstruktur	**BPEL4WS-Konstrukte**
Sequenz	BPEL4WS `sequenz`-Aktivität,d die alle korrespondierenden BPEL4WS Aktivitäten umschließt.
Nebenläufigkeit	BPEL4WS `flow`-Aktivität. Die jeweils die Nebenläufigkeit realisierenden *Step*s werden separat vor bzw. nach der `flow`-Aktivität, in einer zusätzlichen `sequence`-Aktivität eingebettet, abgebildet.
Alternative	BPEL4WS `switch`-Aktivität. Die Bedingungen der unterschiedlichen Zweige entsprechen den `condition`-Bedingungen. Die jeweils die Alternative realisierenden *Step*s werden separat vor bzw. nach der `switch`-Aktivität, in einer zusätzlichen `sequence`-Aktivität eingebettet, abgebildet. Im Falle, dass sich die jeweiligen Bedingungen auf Prozess-*Variable*n oder *Step* Eingaben beziehen, die in Form von Variablen in BPEL4WS abgebildet werden, können diese Bedingungen mit Bezug auf die korrespondierenden Variablen äquivalent im BPDM realisiert werden. Im Falle, dass die *Step*s die Entscheidung selber vornehmen, wird eine zusätzliche Status-Variable eingeführt, die bei Beendigung der korrespondierten BPEL4WS Aktivitäten die Entscheidung zurückliefert und anhand derer die jeweiligen Bedingungen spezifiziert werden.
Schleifen	BPEL4WS `while`-Aktivität. Die `while`-Aktivität kapselt die, den Schleifenkörper enthaltenen *Step*s, korrespondierenden Aktivitäten. Zusätzlich wird in Analogie zur Alternative, jedoch innerhalb der `while`-Aktivität, eine `sequence`-Aktivität eingeführt.

beliebige Kontrollflussstrukturen

Kontrollfluss	**BPEL4WS-Konstrukt**
unstrukturierte Schleifen	Werden von BPEL4WS nicht unterstützt. Siehe hierzu Abschnitt 5.2.5.
unstrukturierte Verzweigungen	Werden von BPEL4WS nicht durch vorhandene Kontrollfluss-Konstrukte (außer `flow`-Aktivität) unterstützt. Siehe hierzu Abschnitt 5.2.5.
unstrukturierte Parallelisierung	Kann in BPEL4WS mit Hilfe der `flow`-Aktivität realisiert werden.

5.2 Limitationen und Komplikationen der Abbildung

Im Kontext dieses Abschnittes werden nachfolgend die bei der Abbildung vorhandenen Limitationen von BPEL4WS und die damit verbundenen Komplikationen näher charakterisiert. Hierzu wird in jedem Abschnitt ein spezifischer Problemfall separat dargestellt, auf den bereits im vorherigen Abschnitt im Kontext der allgemeinen Charakterisierung der Abbildung verwiesen wurde.

5.2.1 abstrakte Prozesse

BPDM als auch BPEL4WS unterstützen abstrakte Prozesse, die zur Beschreibung des öffentlichen Verhaltens eines Prozesses benutzt werden, nicht direkt ausführbar sind und durch andere Prozesse „realisiert“ bzw. „implementiert“ werden und damit vergleichbar zu abstrakten Klassen der objektorientierten Modellierung sind. Aufgrund von Limitationen im Bereich der Struktur von BPEL4WS Prozessen und der syntaktischen Beschreibung der Beziehung zwischen abstrakten und ausführbaren Prozessen in BPEL4WS ergeben sich hinsichtlich der Abbildung von abstrakten BPDM Prozessen auf abstrakte BPEL4WS Prozesse die im Folgenden näher dargestellten Probleme.

Da in BPEL4WS abstrakte Prozesse genau den gleichen Restriktionen bezüglich der Spezifikation der Struktur des Prozessverhaltens wie ausführbare Prozesse unterliegen, müssen diese grundsätzlich mittels einer oder mehrerer „Start-Aktivitäten“, entweder einer `receive`- oder `pick`-Aktivität, die auf den Empfang von Nachrichten warten, initiiert [2] werden. Im BPDM bestehen solche strukturellen Restriktionen für abstrakte Prozesse nicht, wodurch ein abstrakter BPDM Prozess durchaus zu Beginn anhand eines SendMessageTask, der beispielsweise eine Nachricht zum Aufruf einer Operation eines Partners überträgt, initiiert werden kann. Da SendMessageTasks anhand von `invoke`-Aktivitäten in BPEL4WS abgebildet werden und in abstrakten BPEL4WS Prozessen `invoke`-Aktivitäten nicht als Start-Aktivitäten zulässig sind, können aufgrund dieser Limitation abstrakte BPDM Prozesse, die mittels SendMessageTask eingeleitet werden, nicht als abstrakte BPEL4WS Prozesse abgebildet werden.

Ein anderes Problem im Umfeld der Abbildung von abstrakten BPDM Prozesse besteht auf Grund der unterschiedlichen Art und Weise wie abstrakte mit operationalen (ausführbaren) Prozessen im BPDM und BPEL4WS syntaktisch verknüpft werden. Im BPDM erfolgt die Verknüpfung zwischen abstrakten und operationalen Prozessen direkt anhand einer zwischen ihnen im Modell definierten realize-Assoziation. Demgegenüber existiert in BPEL4WS bisher kein vergleichbarer Mechanismus,

[2] in dem Sinne, dass keine anderen Basis-Aktivitäten logisch innerhalb der Definition des Prozessverhaltens vorrangehen

mit dem abstrakte mit den sie implementierenden (ausführbaren) Prozessen in Beziehung zueinander gesetzt werden können. Lediglich anhand der Verwendung derselben PartnerLink Typen und dem Bezug zu den gleichen Rollen im ausführbaren wie im abstrakten Prozess läßt sich indirekt erahnen, welcher abstrakte Prozess durch welchen ausführbaren Prozess implementiert wird. Ein wesentlicher Grund für das Fehlen eines solchen Konstrukts in BPEL4WS ist, dass bisher die Semantik der Implementation Beziehung zwischen abstrakten und ausführbaren Prozessen nicht genau spezifiziert wurde. Damit können zwar abstrakte und sie realisierende BPDM Prozesse in BPEL4WS abgebildet werden; ihre Beziehung, anhand eines syntaktischen Konstrukts zueinander, jedoch nicht.

Ein weiteres Problemfeld stellen kollaborative Prozesse dar.

5.2.2 kollaborative Prozesse

Kollaborative Prozesse werden im BPDM verwendet um die zwischen Partnern geltenden Geschäftsprotokolle aus globaler Sicht zu spezifizieren (siehe hierzu Abschnitt 4.2.1). In BPEL4WS können demgegenüber lediglich abstrakte Prozesse zur Beschreibung von Geschäftsprotokollen verwendet werden, die jeweils genau aus einer Sicht, bezogen auf den zugehörigen Partner, das Geschäftsprotokoll spezifizieren.

Damit ist es prinzipiell nicht möglich kollaborative Prozesse in BPEL4WS durch vorhandene Konstrukte abzubilden, die in äquivalenter Form Geschäftsprotkolle aus globaler Sicht bezüglich allen beteiligten Partnern beschreiben.

Neben kollaborativen Prozessen existieren ebenfalls Probleme bei der Abbildung von Subprozessen.

5.2.3 Subprozesse

Im BPDM besteht die Möglichkeit, dass Prozesse Subprozesse nutzen um einen Teil ihres Verhaltens zu realisieren. Hierbei stellen Subprozesse reguläre Prozesse dar, mit denen der Vater-Prozess auf die gleiche Weise wie mit normalen, gleichgestellten Partner-Prozessen interagiert. Der wesentliche Unterschied hierbei ist, dass der Subprozess Lebenszyklus von dem des Vater-Prozesses abhängig ist. D.h. wird die Vater-Prozess-Instanz beendet, so muss auch die Subprozess-Instanz beendet werden.

BPEL4WS unterstützt das Konzept von Subprozessen für die Geschäftsprozessmodellierung nicht, da grundsätzlich alle miteinander interagierenden Prozesse als eigenständige Prozesse in Form lose gekoppelter Web Services realisiert werden, deren Lebenszyklus voneinander unabhängig ist.

Damit ergibt sich hinsichtlich der Abbildung von Subprozessen in BPEL4WS grundsätzlich ein Problem: Da Subprozesse reguläre Partner-Prozesse im BPDM darstellen, die lediglich im Kontext eines BPDM Prozesses, der mit diesen interagiert, als Subprozesse verwendet werden und damit eine Sonderrolle bzgl. ihres Lebenszyklus besitzen, werden sie als BPEL4WS Prozesse abgebildet, womit speziell im BPEL4WS Prozessmodell nicht explizit erkennbar ist, dass es sich bei dem abgebildeten Partner-Prozess um einen Subprozess handelt. Das in Abbildung 5.1 dargestellte Beispiel verdeutlicht dieses Problem.

`P` ist ein BPDM Prozess der mit dem Prozess `SP` in Form eines Subprozesses über die Interfaces `IP` und `ISP` interagiert, dargestellt anhand eines Connectors `PtoSP` zwischen beiden Interfaces. Beide Prozesse würden in BPEL4WS als Prozesse `P` und `SP` abgebildet, schematisch dargestellt im nachfolgenden Listing, die untereinander anhand eines dem Connector korrespondierenden `partnerLinkType` interagieren. Wobei bzgl. der `partnerLink` Definition im Prozess `P` dieser die Rolle `RoleP` und bzgl. der `partnerLink` Definition im Prozess `SP` die Rolle `RoleSP` einnimmt. Einziges Problem hierbei ist,

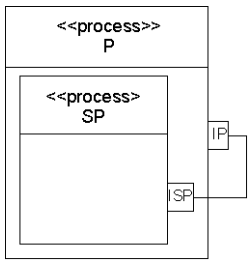

Abbildung 5.1: *Beispiel einer BPDM Prozess, Subprozess Beziehung.*

dass aus den beiden BPEL4WS Prozess Definitionen nicht hervorgeht, dass es sich bei `SP` um einen Subprozess von `P` handelt. Im Gegenteil, `SP` wird als eingenständiger Prozess behandelt, der mit `P` interagiert.

```
<process name="P"...>                              <process name="SP"...>
 ...                                                ....
                <partnerLinkType name="PtoSPPLT">
                  <role myRole="RoleP">
                    <portType name="IPPortType"/>
                  </role>
                  <role partnerRole="RoleSP">
                    <portType name="ISPPortType"/>
                  </role>
                </partnerLinkType>
 ...                                                ...
 <partnerLink name="PtoSPPLT"                      <partnerLink name="PtoSPPLT"
  myRole="RoleP"                                    myRole="RoleSP"
  partnerRole="RoleSP"/>                            partnerRole="RoleP"/>
 ...                                                ...
</process>                                         </process>
```

Daraus folgt, dass die Lebenszyklus Abhängigkeit ebenfalls im BPEL4WS Modell nicht erfasst wird und die BPEL4WS Ausführungsumgebung den Subprozess als normalen Prozess instanziert, der unabhängig vom Lebenszyklus des Vater- in diesem Fall des aufrufenden Partner-Prozesses ist.

Einen zusätzlichen Problembereich bilden Korrelationsprädikate.

5.2.4 Korrelationsprädikate

Korrelationsprädikate werden im BPDM verwendet um eingehende Nachrichten explizit an spezifische Prozessinstanzen weiterzuleiten. Hierzu werden diese den Nachrichten, auf die ein Prozess reagieren soll, zugeordnet. Wird eine Nachricht, für die ein Korrelationsprädikat definiert wurde, von der BPDM Ausführungsumgebung empfangen, wird für alle laufenden Prozessinstanzen dieses Prozesses das Korrelationsprädikat evaluiert, anhand dessen von der Ausführungsumgebung entschieden werden kann, an welche Instanzen die Nachricht weitergeleitet werden muss.

Um die Probleme bei der Abbildung von Korrelationsprädikaten in BPEL4WS besser darstellen zu können, werden Korrelationsprädikate wie folgt induktiv formalisiert: Sei $Rel = \{<, >, =\}$ die Menge möglicher Vergleichsoperationen, $MP_M = \{m_1, \ldots, m_n\}$ die Menge möglicher Nachrichtenteile eine Nachricht M bezüglich der das Korrelationsprädikat definiert wird, $Var = \{var_1, \ldots, var_m\}$ die Menge

an Variablen die im Prozessmodell spezifiziert sind, $Bool = \{\wedge, \vee\}$ die Menge boolscher Operatoren, $Konst = \{c_1, \ldots, c_j\}$ die Menge möglicher Konstanten und $n, m, j \in \mathbb{N}$, dann gilt:

1. $CP_M^{BPDM} = k \circ l$ mit $k \in MP_M$, $l \in Var \cup Konst$ und $\circ \in Rel$ ist ein Korrelationsprädikat bezüglich der Nachricht M.

2. Sind A_M^{BPDM} und B_M^{BPDM} Korrelationsprädikate, so ist auch $A_M^{BPDM} \odot B_M^{BPDM}$ mit $\odot \in Bool$ ein Korrelationsprädikat.

3. Nichts ist ein Korrelationsprädikat, was nicht durch 1. und 2. definiert wurde.

In BPEL4WS kommen ähnlich zu Korrelationsprädikaten Korrelationsmengen zum Einsatz, die anhand einer spezifischen eingehenden oder ausgehenden Nachricht im Laufe des Prozess-Lebenszyklus initialisiert werden und deren Wertausprägung der jeweiligen, sie initialisierenden Prozessinstanz eindeutig zugeordnet sind. Im weiteren Verlauf der Prozessabarbeitung kann beispielsweise bei `receive` Aktivitäten Bezug auf eine initialisierte Korrelationsmenge genommen werden, wodurch beim Empfang einer Nachricht durch die BPEL4WS Ausführungsumgebung überprüft wird, ob die Werte der Korrelationsmenge mit denen der entsprechenden Nachricht übereinstimmen. Ist dies der Fall, wird die Nachricht an genau diese Prozessinstanz weitergeleitet.

Damit die Schwierigkeiten bei der Abbildung von BPDM Korrelationsprädikaten in BPEL4WS besser dargestellt werden können, werden im Folgenden BPEL4WS Korrelationsmengen als Prädikate wie folgt formalisiert: Sei $MProp_M = \{mp_1, \ldots, mp_n\}$, $n \in \mathbb{N}$ die Menge möglicher Nachrichtenteile einer WSDL-Message M, bezüglich der die Korrelationsmenge definiert wird und $P_M = \{p_1, \ldots, p_n\}$ die Menge an Platzhaltern, vergleichbar mit Variablen, für die gilt, dass p_i lediglich den Nachrichtenanteil mp_i einer konkreten Nachricht M^i speichern kann, dann ist eine Korrelationsmenge als Prädikat $CP_M^{BPEL4WS}$ folgendermaßen definiert:

$$CP_M^{BPEL4WS} = \bigwedge_{i=1}^{k} mp_{f(i)} = p_{f(i)}$$

mit $k \in \{1, \ldots, n\}$ und $f : \{1, \ldots, k\} \rightarrow \{1, \ldots, n\}$ mit $f(i) \neq f(j)$ für $i \neq j$ und $n \in$ N. Zusätzlich gelten die folgenden Restriktionen: Alle Platzhalter p_i des Korrelationsprädikats $CP_M^{BPEL4WS}$ werden durch genau eine empfangene oder gesendete Nachricht vom Typ M im Prozess-Lebenszyklus initialisiert. Eine Initialisierung zu einem später Zeitpunkt ist nicht möglich. Dies entspricht dem Verhalten der Initialisierung von Korrelationsmengen.

Anhand beider formalen Definitionen lassen sich, aufgrund zahlreicher Beschränkungen der äquivalenten BPEL4WS Korrelationsprädikate, folgende Probleme für die Abbildung von BPDM Korrelationsprädikaten in BPEL4WS identifizieren:

1. BPDM bietet die Möglichkeit Konstanten c_i zur Definition von Korrelationsprädikaten zu verwenden, BPEL4WS demgegenüber lediglich die Möglichkeit der Spezifikation von Platzhaltern p_i, die im Gegensatz zu Konstanten anhand von Nachrichtenteilen initialisiert werden und damit nicht den Charakter einer Konstante besitzen.

2. Platzhalter p_i in BPEL4WS korrespondierenden Korrelationsprädiakten sind vergleichbar mit den in BPDM-Korrelationsprädikaten verwendbaren Variablen var_i. Ein wesentlicher Unterschied zwischen beiden besteht darin, dass alle Platzhalter des korrespondierenden Korrelationsprädikats durch eine Nachricht vom Typ M initialisiert werden müssen. Dagegen können im BPDM Variablen von unterschiedlichen Nachrichten mit Werten belegt werden.

3. Die in BPDM Korrelationsprädikaten verwendeten Variablen können zu unterschiedlichen Zeitpunkten im Prozess-Lebenszyklus mit anderen Werten anderer Nachrichten belegt werden. Demgegenüber können bei BPEL4WS korrespondierenden Korrelationsprädikaten die Platzhalter p_i lediglich genau einmal mit Werten einer bestimmten Nachricht belegt werden.

4. In BPDM Korrelationsprädikaten kann mehrmals auf den gleichen Nachrichtenteil m_i Bezug genommen werden, beispielsweise: $CP_M^{BPDM} = (m_i = var_i) \vee (m_i = var_j)$. Dagegen kann in BPEL4WS Korrelationsprädiakten ein Nachrichtenteil mp_i nur genau einmal vorkommen.

5. Im BPDM können die Vergleichsoperatoren $<, >, =$ zur Spezifikation von Korrelationsprädikaten verwendet werden. In BPEL4WS korrespondierenden Korrelationsprädikaten lediglich $=$.

6. Die einzelnen Vergleichsterme können im BPDM entweder mittels $\wedge$ oder $\vee$ verknüpft werden. In BPEL4WS lediglich anhand von $\wedge$.

7. Aufgrund der Punkte 1. und 5. besteht die Möglichkeit, dass Nachrichten an mehrere Prozessinstanzen weitergeleitet werden. In BPEL4WS ist dies anhand der eindeutigen Zuordnung von Korrelationsmengen zu Prozessinstanzen nicht möglich.

Weitere Komplikationen sind im Bereich der Abbildung des Kontrollflusses von BPDM Prozessen zu finden.

5.2.5 Kontrollfluss

Bei der Abbildung von BPDM Prozessen spielt die Realisierung des durch sie spezifizierten Kontrollflusses in BPEL4WS, neben der Charakterisierung der verschiedensten Modellelemente, eine zentrale Rolle. Da der Kontrollfluss im BPDM sehr flexibel, ohne strukturelle Restriktionen bezüglich der daraus resultierenden Kontrollflussstrukturen, spezifizierbar ist, existieren im Umfeld der Abbildung des Kontrollflusses in BPEL4WS Probleme, die im Folgenden näher charakterisiert werden. Hierbei wird zu Beginn das Problem der Abbildung unstrukturierter Schleifen und abschließend das Problem der Abbildung unstrukturierter Verzweigungen dargestellt.

unstrukturierte Schleifen

Schleifen stellen grundlegende Kontrollflussstrukturen dar, die zur Beschreibung von Wiederholungen einzelner Arbeitsschritte innerhalb eines Geschäftsprozessmodells verwendet werden. Hierbei muss grundsätzlich zwischen zwei unterschiedlichen Arten von Schleifen unterschieden werden: strukturierte und unstrukturierte Schleifen. Der wesentliche Unterschied zwischen beiden besteht in der Anzahl von Kanten, die den Schleifenkörper verlassen (Austrittspfade bzw. Austrittspunkte) bzw. in ihn münden (Eintrittspfade bzw. Eintrittspunkte). So besitzen Schleifenkörper strukturierter Schleifen lediglich eine eingehende und eine ausgehende Kante. Bei unstrukturierten Schleifen dagegen verfügen sie über eine flexible Anzahl eingehender und ausgehender Kanten[3]. Dies ist anhand der nachfolgenden schematischen Abbildung strukturierter und unstrukturierter Schleifen verdeutlicht. Die gestrichelten Rechtecke stellen hierbei beliebige Arbeitsschritte, die Diamantsymbole Verzweigungs- (Decision) bzw. Verschmelzungs (Merge)-Knoten und die schwarzen Punkte jeweils Eintritts- bzw. Austrittspunkte in bzw. aus dem Schleifenkörper, dar.

Vom BPDM werden prinzipiell unstrukturierte Schleifen, damit verbunden auch strukturierte Schleifen, durch die Möglichkeit der Rückführung einzelner Kontroll- bzw. Datenkanten zu vorherigen im Kontrollfluss spezifizierten Tasks unterstützt. Die folgende Abbildung zeigt ein abstraktes Beispiel einer im BPDM modellierten unstrukturierten Schleife. Der grau hinterlege Bereich stellt hierbei den

[3]Es wird hier davon ausgegangen, dass alle eingehenden und ausgehenden Kanten nicht parallelisiert sind.

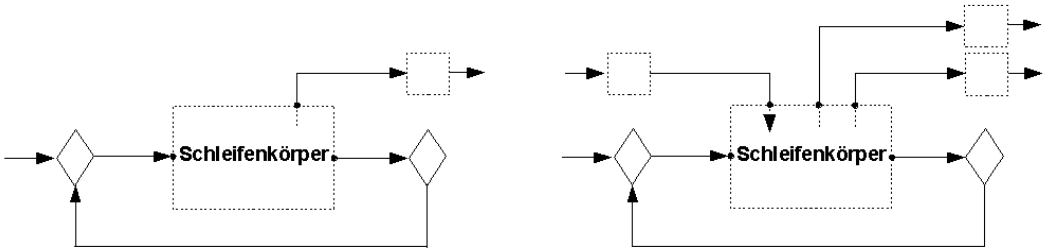

Abbildung 5.2: *Schema strukturierter und unstrukturierter Schleifen.*

Schleifenkörper dar, der in diesem Fall zwei alternative Austrittspfade (B→C, D→E) besitzt.

BPEL4WS bietet demgegenüber ausschließlich die Möglichkeit strukturierte Schleifen zur Beschreibung von Wiederholungen einzelner Aktivitäten anhand von `while`-Aktivitäten zu verwenden, die lediglich über einen klar definierten Eintritts- bzw. Austrittspunkt betreten bzw. verlassen werden und damit über keinerlei Mechanismus zur Definition alternativer Eintritts- und Austrittspunkte verfügt.

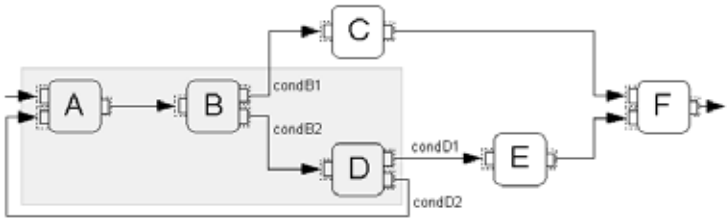

Abbildung 5.3: *Abstraktes Beispiel einer unstrukturierten Schleife im BPDM.*

Damit können unstrukturierte Schleifen in BPEL4WS nicht ohne weiteres abgebildet werden und es stellt sich die Frage, ob überhaupt unstruktierte Schleifen in adäquater Form mit Hilfe strukturierter Schleifen abgebildet werden können und wie eine solche Abbildung realisiert werden kann.

unstrukturierte Verzeigungen

Wie im vorherigen Abschnitt über Schleifen kann auch bei Verzweigungen zwischen zwei unterschiedlichen Arten unterschieden werden: strukturierte (blockorientierte) und unstrukturierte Verzweigungen. Der wesentliche Unterschied zwischen beiden besteht darin, dass bei strukturierten Verzweigungen alle spezifizierten alternativen Zweige, die einen Verzweigungsknoten verlassen, durch einen korrespondierenden Verschmelzungsknoten zusammengeführt werden, wohingegen bei unstrukturierten Verzweigungen keine Restriktionen bezüglich der Struktur der Verzweigung existieren und beliebige Zweige durch Verschmelzungs-Knoten zusammengeführt werden können. Die folgende Abbildung zeigt die schematische Darstellung einer strukturierten und unstrukturierten Verzweigung.

Da von BPEL4WS unstrukturierte Verzweigungen mittels der `flow`-Aktivität untersützt werden, können damit die im BPDM spezifizierten unstrukturierte Verzweigungen (Abbildung 5.5) prinzipiell in BPEL4WS abgebildet werden. Da zur Realisierung der Abbildung des BPDM Kontrollflusses in

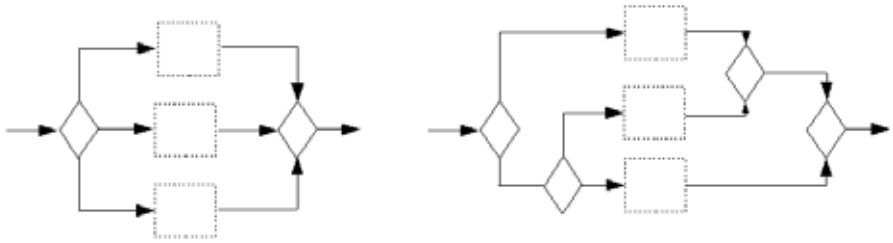

Abbildung 5.4: *Schema einer strukturierten und unstrukturierten Verzweigung.*

BPEL4WS im Folgenden lediglich die bestehenden blockorientierten Konstrukte, aufgrund besserer Strukturiertheit, verwendet werden sollen, besteht damit die Schwierigkeit unstrukturierte Verzweigungen in BPEL4WS in geeigenter Art und Weise abzubilden.

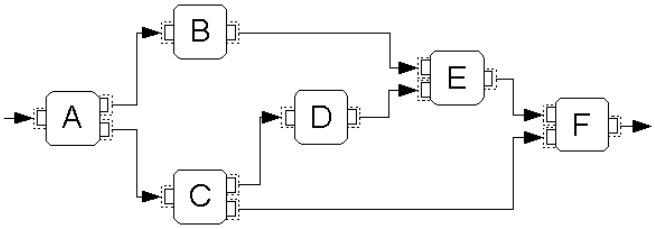

Abbildung 5.5: *Abstraktes Beispiel einer unstrukturierten Verzweigung im BPDM.*

5.2.6 Datenfluss - Multiple Daten Output- und Input-Pins

Im Zusammenhang mit multiplen Input- und Output-Pins wurden im vorherigen Abschnitt speziell die damit verbundenden Probleme im Bereich der Abbildung möglicher Kontrollflussstrukturen in BPEL4WS aufgezeigt. In diesem Abschnitt soll ein weiteres damit verbundenes Problem dargestellt werden, welches im Kontext der Realisierung des Datenflusses, speziell von mehreren Daten Output- und Input-Pins, vorhanden ist und bei der Abbildung von Tasks mittels `invoke`-Aktivitäten in BPEL4WS auftritt.

Im BPDM werden Eingabe- bzw. Ausgabedaten über Tasks zugeordnete Daten Input- und Output-Pins zu bzw. von ihnen abgeführt. Hierbei können Tasks nicht nur über einen Input- bzw. Output-Pin verfügen, sondern über mehrere, die entweder gruppiert oder ungruppiert sind, so dass entweder über alle in einer Gruppe enthaltenen Pins (implizite UND Semantik gruppierter Pins) oder ein einzelnes ungruppiertes Pin (implizite ODER Semantik von Pins und Gruppen) Token empfangen bzw. emittiert werden.

In BPEL4WS werden `invoke`-, `receive`- und `reply`-Aktivitäten Daten in Form einer typisierten Variable übergeben bzw. von ihn zurückgeliefert, wodurch prinzipiell keine Möglichkeit besteht, dass alternative Ein- und Ausgabedaten den Aktivitäten übergeben bzw. von ihnen zurückgeliefert werden.

Damit stellt sich die Schwierigkeit, wie, wenn Tasks anhand von `invoke`-Aktivitäten abgebildet werden, die Eingabe- und Ausgabedaten, die über verschiedene Pins ein- bzw. zurückgegeben werden, in BPEL4WS geeignet abgebildet werden können.

Neben Problemen im Bereich des Daten- und Kontrollflusses treten zusätzliche Schwierigkeiten im Bereich der Abbildung von Repositories, ForEach-Tasks, manuellen Tasks und Ressourcen auf.

5.2.7 Repository

Repositories stellen einen permanenten Datenspeicher für eine vordefinierte Anzahl von Token gleichen Typs zur Verfügung, vergleichbar einer Datenbank, auf die unterschiedliche Prozesse während ihrer Laufzeit entweder mittels Copy- oder Remove-Semantik zugreifen können.

In BPEL4WS können zur Datenhaltung lediglich typisierte Variablen eingesetzt werden, die lokal, bezüglich der zugehörigen Prozessinstanz, verwendet werden und auf die von anderen Prozessen nicht zugegriffen werden kann. Darüberhinaus geht der Variableninhalt nach der Terminierung der Prozessintanz verloren.

Damit ergibt sich prinzipiell das Problem, dass auf Grund der Externalität und Persistenz sowie der Copy- und Remove-Semantik von Repositories diese nicht mit den Standardkonzepten zur Datenhaltung in Form von Variablen in BPEL4WS abgebildbar sind, da zum einen Variablen nicht global verfügbar und persistent sind, sowie der Zugriff ausschließlich anhand der Copy-Semantik durchgeführt wird. An dieser Stelle sei bemerkt, dass es sich hierbei nicht um ein Problem im eigentlichen Sinne handelt, da die Persistens von Daten über Prozesse hinweg ausserhalb des Gültigkeitsbereichs von BPEL4WS liegt. Es stellt sich lediglich die Frage, wie die Benutzung externer Repositories in BPEL4WS modelliert werden kann.

5.2.8 ForEach-Task

ForEach-Task sind strukturierte Task, die eine Menge von Token in Form einer Kollektion als Eingabe bekommen und für jedes Token der Kollektion die eingebetteten Tasks (ForEach-Block) entweder seriell oder parallel ausführen. D.h. entweder wird der ForEach-Block für jedes Token nacheinander ausgeführt (seriell) oder für alle Token werden gleichzeitig mehrere ForEach-Blöcke gestartet, die parallel nebeneinander ablaufen. Jedes vom ForEach-Block verarbeitete Token wird an genau die gleiche Stelle in der Ausgabe Kollektion emittiert, an der es auch in der Eingabe-Kollektion vorhanden war. Die Ausführung des ForEach-Tasks ist beendet, wenn für alle Token der Kollektionen der ForEach-Block ausgeführt wurde.

In BPEL4WS besteht zwar die Möglichkeit die im BPDM vorhandenen Kollektionen in Form von Variablen, die anhand von XML Schema `element`-Definitionen typisiert sind, abzubilden, bei denen das `maxOccures`-Attribut die Anzahl an Einträgen der Kollektion spezifiziert und der Typ der Elemente dem Typ der Token in der Kollektion entspricht, aber eine BPEL4WS-Aktivität, die entweder seriell oder parallelisiert die einzelnen Elemente durch eingebettete Aktivitäten verarbeiten kann, existiert als solches nicht. Daher besteht die Schwierigkeit den ForEach-Task, sowohl mit serieller als auch paralleler Semantik in BPEL4WS, mit den zur Verfügung stehenden Konstrukten in geeigneter Art und Weise zu repräsentieren.

5.2.9 manuelle Tasks, Ressourcen

BPDM ist konzipiert um Geschäftsprozesse, die innerhalb eines Unternehmens ablaufen und mit anderen Geschäftsprozessen interagieren, zu beschreiben. Hierbei können Aktivitäten entweder komplett automatisiert oder mit Hilfe menschlichen Einflusses durchgeführt werden, was anhand von Ressourcenmodellen spezifiziert wird.

BPEL4WS als Prozessbeschreibungssprache bietet bis zum jetzigen Zeitpunkt (Version 1.1) keine Unterstützung zur Modellierung durch Menschen auszuführender Aktivitäten, wodurch aufgrund dieser Limitation gegenüber BPDM die Schwierigkeit besteht manuelle Aktivitäten in BPDM Prozessen in BPEL4WS geeignet abzubilden. Darüberhinaus stellt sich damit das Problem, wie in BPEL4WS die in Ressourcenmodellen spezifizierten Anforderungen von Tasks repräsentiert werden können.

Da aufgrund der hier charakterisierten Limitationen und Komplikationen die Abbildung von BPDM Prozessen in BPEL4WS zum Teil sehr stark eingeschränkt wird und das Ziel besteht so viel wie möglich Sprachelemente in BPEL4WS geeignet zu repräsentieren, werden in den folgenden Kapiteln für die hier aufgezeigten Komplikationen mögliche Lösungsansätze charakterisiert, die in Teilen auf den bereits im Kontext der BPEL4WS Issues [12] gemachten Lösungs- und Erweitungsvorschlägen basieren.

Kapitel 6

Darstellung möglicher Lösungsansätze

Dieses Kapitel beschreibt im Folgenden für die im vorherigen Kapitel aufgezeigten Limitationen und Komplikationen Lösungsansätze, die bereits die im Kontext der BPEL4WS Issue Liste [12] gemachten möglichen Erweiterungen zukünftiger BPEL4WS Versionen mit berücksichtigen. Hierbei erfolgt die Darstellung eines möglichen Ansatzes zur Realisierung des von BPDM Prozessen spezifizierten Kontrollflusses in BPEL4WS aufgrund dessen Komplexität und Umpfangs gesondert in Kapitel 7.

6.1 abstrakte Prozesse

Aufgrund der in Abschnitt 5.2.1 beschriebenen Limitationen bezüglich der Struktur von abstrakten Prozessen und der syntaktischen Beschreibung der Beziehung zwischen abstrakten und ausführbaren Prozessen in BPEL4WS ergeben sich zwei wesentliche Probleme für die Abbildung abstrakter BPDM Prozesse: zum einen, dass aufgrund der strukturellen Restriktion von BPEL4WS Prozessen abstrakte BPDM Prozesse, die mit einem SendMessageTask initiiert werden, nicht in BPEL4WS repräsentiert werden können und zum anderen, dass die zwischen abstrakten und ausführbaren BPDM Prozessen spezifizierte realize-Assoziation in BPEL4WS nicht abgebildet werden kann.

Bezüglich des ersten Problems gibt es zur Zeit die Diskussion, dass die Beschränkung der Initiierung von BPEL4WS Prozesse ausschließlich auf `receive`- und `pick`-Aktivitäten im Kontext der Spezifikation von abstrakten Prozessen und damit der Beschreibung von Geschäftsprotokollen keinen Sinn macht, da das zwischen Partnern definierte Geschäftsprotokoll von mindestens einem Partner durch das Versenden einer Nachricht initiiert wird und damit anhand der Beschreibung von abstrakten BPEL4WS Prozesses mindestens einer dieser Prozesse durch eine `invoke`-Aktivität eingeleitet werden muss und der Aspekt, wie der initiierende Prozess gestartet wird, ein Implementationsdetail darstellt, welches nicht vom abstrakten BPEL4WS Prozess spezifiziert werden muss, solange dies nicht Gegenstand des zugrundeliegenden Geschäftsprotokolls ist [40]. Dies soll anhand der Beschreibung des in Abbildung 6.1 als UML Sequenzdiagramm dargestellten Geschäftsprotokolls einer einfachen Käufer-Verkäufer Kollaboration in BPEL4WS verdeutlicht werden.

Ein Käufer sendet zu Beginn der Kollaboration eine synchrone *Produkte* Nachricht mit der er alle verfügbaren Produkte des Verkäufers erfragt. Anschließend wird vom Käufer eine asynchrone Nachricht zur Bestellung eines Produkts verschickt der daraufhin mit dem versenden einer *Bestellbestättigung* als asynchrone Nachricht antwortet.

Da BPEL4WS über keinerlei Möglichkeit verfügt Geschäftsprotokolle aus globaler Sicht zu beschreiben (Abschnitt 5.2.2), werden diese anhand eines, jeder beteiligten Rolle, zugeordneten abstrakten

Abbildung 6.1: *Geschäftsprotokoll einer einfachen Käufer-Verkäufer Kollaboration.*

Prozesses beschrieben, der das lokale Verhalten der jeweiligen Rolle bezüglich des Geschäftsprotokolls erfasst. Das folgende Listing zeigt schematisch den abstrakten Prozess für die Verkäufer Rolle:

```
<process abstractProcess="true"...>
...
    <sequence>
        <receive operation="Produkte"             variable="..." />
        <reply   operation="Produkte"             variable="..." .../>
        <receive operation="bestellProdukt"       variable="..." .../>
        <invoke  operation="Bestellbestaetigung"  inputVariable="..." .../>
    </sequence>
</process>
```

Wie man erkennt, wird in Analogie zum Sequenzdiagramm auf Seiten des Verkäufers auf das Eintreffen der *Produkte* Nachricht gewartet, die mittels der `reply`-Aktivität mit einer Liste von möglichen Produkten beantwortet wird. Anschließend wird auf die Übermittlung der *bestellProdukt* Nachricht gewartet, die mit Hilfe einer asynchronen Übertragung der *Bestellbestätigung* Nachricht bestätigt wird. Demgegenüber hätte der abstrakte Prozess für die Käufer Rolle die folgende Gestalt:

```
<process abstractProcess="true"...>
...
    <sequence>
         <invoke  operation="Produkte"             inputVariable="..."
                  outputVariable="..." .../>
         <invoke  operation="bestellProdukt"       inputVariable="..." .../>
         <receive operation="Bestellbestaetigung" variable="..." .../>
    </sequence>
</process>
```

Da dieser abstrakte Prozess nicht der BPEL4WS Spezifikation, die festlegt, dass jeder Prozess mit einer Start-Aktivität eingeleitet werden muss, entspricht, kann dieses Geschäftsprotokoll nur unvollständig in BPEL4WS spezifiziert werden. Dies gilt grundsätzlich für alle Rollen, die an einem Geschäftsprotokoll beteiligt sind und zu Beginn eine Nachricht senden anstatt auf den Empfang einer Nachricht zu warten.

Im Kontext der BPEL4WS Issue Liste (Issue 99) werden für diese Limitation mittlerweile verschiedene Lösungsvorschläge diskutiert, die diese Restriktion von BPEL4WS versuchen zu beseitigen. Ein Vorschlag [40] sieht die Einführung aller Basis-Aktivitäten, damit auch `invoke`, als Start-Aktivitäten für abstrakte BPEL4WS Prozesse vor, von denen `terminate` und `reply` als Start-Aktivitäten ausgeschlossen sind, da `terminate`-Aktivitäten lediglich ausführbaren Prozessen vorbehalten sind und `reply`-Aktivitäten korrespondierende `receive`-Aktivitäten benötigen. Ein anderer Vorschlag führt sog. abstrakte Prozessfragmente [18] ein, die den gleichen syntaktischen als auch semantischen Regeln

wie abstrakte Prozesse unterliegen, mit der Außnahme, dass diese keine Start-Aktivitäten enthalten und keine Information bzgl. deren Initialisierung spezifizieren. Hierbei werden abstrakte Prozesse und abstrakte Prozessfragmente anhand eines zusätzlichen `abstractFragment`-Werts des `isAbstract`-Attributs unterschieden.

Speziell für die Abbildung abstrakter BPDM Prozesse in abstrakte BPEL4WS Prozesse ist hierbei der erste Vorschlag zu bevorzugen, da dieser es ermöglicht einleitende SendMessageTasks in äquivalente `invoke`-Aktivitäten abzubilden, anstatt diese unter Verwendung des zweiten Vorschlages nicht mit berücksichtigen zu können.

Für das Problem der Abbildung der realize-Assoziation zwischen abstrakten und operationalen BPDM Prozessen in BPEL4WS kann als möglicher Lösungsvorschlag lediglich eine BPEL4WS Extension spezifiziert werden, mittels derer abstrakte und ausführbare BPEL4WS Prozesse, auf syntaktischer Ebene, in Beziehung zueinander gesetzt werden können, da keine äquivalente semantische Repräsentation mit den vorhandenen Sprachelementen möglich ist. Diese könnte beispielsweise wie folgt aussehen:

```
<process isAbstract="false"
         ext:implementedAbstractProcesses name="list of qname" ...>
</process>
```

Hierbei spezifiziert das `implementedAbstractProcesses`-Attribut eine Liste abstrakter Prozesse anhand ihrer vollständigen qualifizierenden Namen, die durch den ausführbaren Prozess implementiert werden, wodurch speziell die realize-Assoziationen in BPDM Prozessmodellen in Form des `implementedAbstractProcesses`-Attributes in BPEL4WS abgebildet werden können.

6.2 ForEach Task

Wie bereits im vorherigen Kapitel dargelegt, besteht die Schwierigkeit, dass in BPEL4WS kein vergleichbares Konstrukt zum ForEach-Task, mit dem eine Kollektion von Daten durch eine Menge von Steps (ForEach-Block), entweder parallel oder seriell, abgearbeitet werden kann, vorhanden ist. Dieser Abschnitt stellt daher zwei mögliche Ansätze vor, die zur Abbildung von ForEach-Tasks in BPEL4WS eingesetzt werden können.

Da bis zum jetzigen Zeitpunkt Kollektionen als auch das Verhalten von ForEach-Tasks bezüglich neu in die Kollektion aufzunehmender Token im konzeptuellen Metamodell nicht weiter spezifiziert wurden, werden an dieser Stelle für eine genauere Charakterisierung möglicher Realisierungen des ForEach-Tasks in BPEL4WS zusätzliche Annahmen gemacht, die aufgrund der Realisierung von ForEach-Tasks mittels `ExpansionRegion`s im BPDM von deren Eigenschaften abgeleitet werden. D.h. eine Kollektion kann lediglich eine maximale Anzahl T an Token aufnehmen. Die Ausführung des ForEach-Tasks erfolgt, sobald Token in der Kollektion vorhanden sind.

Um einen besseren Einblick in die hier vorgestellten Ansätze zu geben, werden diese zusätzlich anhand des in Abbildung 6.2 gezeigten *Prüfe Bestellungen* ForEach-Task demonstriert. Dieser besitzt eine Eingabekollektion, die maximal T Token vom Typ *Bestellung* aufnehmen kann, einen gekapselten Task *Prüfe Bestellung*, der jeweils eine in der Kollektion enthaltene *Bestellung* auf Vollständigkeit überprüft, sowie eine Ausgabekollektion, in der die geprüften Bestellungen vom Typ *gepBestellung* emittiert werden.

Die erste Schwierigkeit bei der Abbildung des ForEach-Tasks besteht darin die jeweiligen Ein- und Ausgabekollektionen sowie die Kollektionen mit Pins verbindenden Datenkanten in BPEL4WS abzubilden. Wie bereits in Abschnitt 5.2.8 beschrieben, können Kollektionen anhand von Variablen in Analogie zu Pins, die durch WSDL-Messages mit einem `part`, der anhand einer XML Schema `element`-Definition, die mittels des `maxOccurs=T` Attributs die maximale Anzahl an Einträgen der Kollektion spezifiziert, typisiert ist, abgebildet werden. Die jeweils eine Kollektion und ein Pin verbindenden Datenkanten werden anhand der `assign`-Aktivität abgebildet, die im Falle, dass die Quelle

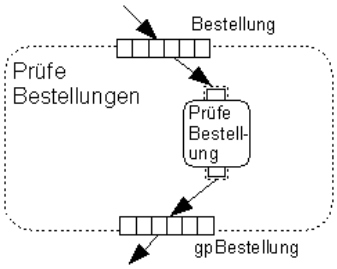

Abbildung 6.2: *Beispiel ForEach-Task Prüfe Bestellungen.*

eine Kollektion und das Ziel ein Pin ist, ein Element mittels einer Erweiterung der BPEL4WS `from`- und `to`-Ausdrücke in die jeweilige, dem Pin korrespondierende Variable bzw. im umgekehrten Fall von der Variable in die Kollektions-Variable kopiert. Die Spezifikation des jeweils gewünschten Elements einer Kollektions-Variable kann hierbei anhand eines XPath-Ausdrucks der Form `/name[index]` erfolgen, wobei `name` dem Namen des zur Spezifikation der Kollaboration verwendeten XML Schema Elements entspricht.

Das folgende Listing zeigt die Repräsentation der Ein- und Ausgabekollektion des *Prüfe Bestellung* ForEach-Tasks. Die Eingabekollektion wird anhand der `BestellungKollektion`-Variable, die Ausgabekollektion mittels `gpBestellungKollektion`-Variable charakterisiert, die jeweils anhand der `BestellungKollektionMsg`- und `gpBestellungKollektionMsg`-Nachricht typisiert sind, deren `parts` `Bestellungen` und `gpBestellungen`, die Kollektionen, anhand der Möglichkeit mehrere Elemente vom XML Schema Typ `BestellungType` bzw. `gpBestellungType` aufzunehmen, charakterisiert.

```
<element name="BestellungKElement"    minOccurs="1" maxOccurs="T" type="BestellungType"/>
<element name="gpBestellungKElement"  minOccurs="1" maxOccurs="T" type="gpBestellungType"/>

<message name="BestellungKollektionMsg"/>
    <part name="Bestellungen" element="BestellungKElement"/>
</message>

<message name="gpBestellungKollektionMsg"/>
    <part name="gpBestellungen" element="gpBestellungKElement"/>
</message>
...
<variables>
    <variable name="BestellungKollektion"   message="BestellungKollektionMsg"/>
    <variable name="gpBestellungKollektion" message="gpBestellungKollektionMsg"/>
</variables>
```

Die Darstellung der Abbildung von Kollektionen mit Pins verbindenden Datenkanten erfolgt in einem der nächsten Listings.

Die zweite Schwierigkeit liegt in der Realisierung der Ausführungssemantik, entweder seriell oder parallel, von ForEach-Tasks.

Der serielle Fall kann in BPEL4WS relativ einfach unter Verwendung der `while`-Aktivität und einer Funktion zum Zählen der Elemente der Eingabekollektions-Variablen, beispielsweise die XPath `count`-Funktion, realisiert werden. Hierzu werden die durch den ForEach-Task gekapselten Steps je-

weils in korrespondierende BPEL4WS Aktivitäten, die in einer `scope`-Aktivität[1] eingebettet und von der `while`-Aktivität umrahmt werden, abgebildet. Zusätzlich wird anhand der `count`-Funktion die Anzahl der in der Kollektions-Variable enthaltenen Elemente bestimmt, die iterativ durch die `while`-Aktivität von der dem ForEach-Block korrespondierenden `scope`-Aktivität verarbeitet werden.

Das nachfolgende Listing zeigt die Realisierung anhand des hier vorgestellten Beispiels für den Fall, dass der ForEach-Task seriell ausgeführt wird. Es werden für die `while`-Aktivität eine Zähl-Variable `InputCounter` und eine Variable `InputCount`, die die Anzahl der Elemente der Eingabekollektion `BestellungKollektion` aufnimmt, definiert. Diese werden zu Beginn anhand der `assign`-Aktivität mit 0 bzw. der Anzahl der Elemente der Eingabekollektion belegt. Der ForEach-Task wird als `while`-Aktivität mit einer `scope`-Aktivität, die den ForEach-Block repräsentiert, abgebildet, der *Prüfe Bestellung*-Task anhand einer `invoke`-Aktivität und zusätzlich den Input- und Output-Pins korrespondierenden Variablen `PruefTaskInputVar` und `PruefTaskOutputVar` und die jeweils die Pins mit den Kollektion verbindenden Datenkanten mittels des in einem der vorherigen Absätze beschriebenen Mechanismus durch jeweils eine `assign`-Aktivität innerhalb der `scope`-Aktivität.

```
<message name="BestellungMsg">
    <part name="Bestellung" type="BestellungType"/>
</message>
<message name="gpBestellungMsg">
    <part name="gpBestellung" type="gpBestellungType"/>
</message>

<variables>
    <variable name="InputCount"    type="int"/>
    <variable name="InputCounter"  type="int"/>
</variable>
...
<assign>
    <copy><from variable="BestellungKollektion" part="Bestellungen"
                  query="count('/BestellungKElement')"/>
          <to variable="InputCount"/></copy>
    <copy><from expression="0"/>
         <to variable="InputCounter"/></copy>
</assign>
<while condition="bpws:getVariableData(InputCounter)<bpws:getVariableData(InputCount)">
   <scope name="PruefeBestellungen"><sequence>
       <variables>
           <variable name="PruefTaskInputVar"   message="BestellungMsg"/>
           <variable name="PruefTaskOutputVar"  message="gpBestellungMsg"/>
       </variables>
       <assign>
           <copy><from variable="BestellungKollektion" part="Bestellungen"
                       query="/BestellungKElement[InputCounter]"/>
                 <to variable="PruefTaskInputVar" part="Bestellung"/></copy>
        </assign>
        <invoke ... inputVariable="PruefTaskInputVar" outputVariable="PruefTaskOutputVar"/>
        <assign>
            <copy><from variable="PruefTaskOutputVar"   part="gpBestellung"/>
                  <to variable="gpBestellungKollektion" part="gpBestellungen"
                      query="/gpBestellungKElement[InputCounter]"/></copy>
            <copy>
```

[1] Die `scope` Aktivität wird deshalb verwendet, da es sich bei ForEach-Tasks um strukturierte Tasks handelt, siehe hierzu Abschnitt 5.1.3, Abbildung von *StructuredTask*.

```
                <from expression="bpws:getVariableData('InputCounter')+1"/>
                <to variable="InputCounter"/></copy>
        </assign>
    </sequence></scope>
</while>
```

Der parallelisierte Fall ist gegenüber dem serialisierten komplexer, da die Parallelisierung nur anhand der `flow`-Aktivität möglich ist, bei der bereits zur Entwurfszeit die Anzahl der zu parallelisierenden Aktivitäten bekannt sein muss, die sich aber bei parallelisierten ForEach-Tasks unter den zu Beginn gemachten Annahmen während der Ausführungszeit ändern kann.

Ein möglicher Ansatz zur Realisierung parallelisierter ForEach-Tasks besteht darin, mittels der Abbildung des ForEach-Blocks als eigenständigen BPEL4WS Prozess (ForEach-Prozess), der anhand einer Prozess-Schnittstellenoperation (hier `startForEachProcess`), die lediglich ein Element der Kollektion als Übergabeparameter besitzt und anhand einer One-way `invoke`-Aktivität aufgegrufen wird, den ForEach-Task mittels einer `while`-Aktivität, die in Analogie zum seriellen Fall, für jedes Element der Kollektion eine ForEach-Prozessinstanz startet, abzubilden, da aufgrund der Verwendung von One-way `invoke`-Aktivitäten nicht bis zur Beendigung der einzelnen Prozessinstanzen gewartet, sondern diese quasi parallel ausgeführt werden. Einziges Problem hierbei ist die Frage, wie die einzeln ablaufenden ForEach-Prozessinstanzen synchronisiert werden können, so dass die reguläre Abarbeitung erst fortgesetzt wird, wenn alle parallelen Ausführungen beendet sind. Hierzu kann eine Callback-Operation auf Seiten des Prozesses, der die ForEach-Prozesse aufruft, zusätzlich spezifiziert werden (hier `CallBackForEachProcess`), die von den ForEach-Prozessen kurz vor ihrer Beendigung mittels einer One-way `invoke`-Aktivität aufgerufen wird und anhand derer zusätzlich das durch den ForEach-Prozess bearbeitete Kollektionselement übergeben wird. Dadurch kann im Prozess durch eine `receive`-Aktivität, die mit der Anzahl der gestarteten ForEach-Prozesse anhand einer `while`-Aktivität wiederholt ausgeführt wird, auf den Aufruf dieser Operation gewartet werden, so dass die `while`-Aktivität erst beendet wird, wenn alle ForEach-Prozesse beendet wurden. Hierfür ist es notwendig einen zusätzlichen Zähler einzuführen, der beim Empfang eines solchen Aufrufs innerhalb der `while` Aktivität erhöht und mit der Anzahl der gestarteten ForEach-Prozesse als `while`-Bedingung verglichen wird.

Das nachfolgende Listing zeigt die Realisierung anhand des hier verwendeten Beispiels für den Fall der parallelisierten Abarbeitung des ForEach-Task. Wie für den seriellen Fall ist eine `while`-Aktivität spezifziert, die diesmal mittels einer One-way `invoke`- Aktivität die ForEach-Prozesse startet. Da hierbei keine Ausgabewerte zurückgeliefert werden, wird anschließend lediglich der Zähler `InputCounter` für die bisher gestarteten ForEach-Prozesse um eins erhöht. Nach erfolgreichem Starten aller Prozesse wird der `InputCounter` mittels einer `assign`-Aktivität auf 0 gesetzt und es wird anhand der nachfolgenden `while`-Aktivität auf die Beendigung alle ForEach-Prozesse *Prüfe Bestellung* durch die `receive`-Aktivität gewartet, die erst verlassen wird, sobald alle Prozesse durch den Aufruf der Prozessschnittstellen-Operation ihre Beendigung signalisiert haben.

```
...
<while condition="bpws:getVariableData(InputCounter)<bpws:getVariableData(InputCount)">
    <scope>
        <variables>
           <variable name="ForEachProcInputVar" message="BestellungMsg"/>
        </variables>
        <sequence>
            <assign>
                <copy><from variable="BestellungKollektion" part="Bestellungen"
                            query="/BestellungKElement[InputCounter]"/>
                      <to variable="ForEachProcessInputVar" part="Bestellung"/></copy>
            </assign>
            <invoke operation="startForEachProcess" inputVariable="ForEachProcInputVar".../>
            <assign>
```

```
                <copy><from expression="bpws:getVariableData('InputCounter')+1"/>
                      <to variable="InputCounter"/></copy>
                </assign>
        </sequence>
    </scope>
</while>
<assign>
    <copy><from expression="0"/>
          <to variable="InputCounter"/></copy>
</assign>
<while condition="bpws:getVariableData(InputCounter)<bpws:getVariableData(InputCount)">
<scope>
    <variables>
            <variable name="ForEachProcOutputVar"  message="gpBestellungMsg"/>
        </variables>
        <sequence>
            <receive operation="CallBackForEachProcess" variable="ForEachProcOutputVar".../>
             <assign>
                <copy><from variable="ForEachProcessOutputVar"/>
                      <to variable="gpBestellungKollektion" part="gpBestellungen"
                          query="/gpBestellungKElement[InputCounter]"/></copy>
                <copy><from expression="bpws:getVariableData('InputCounter')+1"/>
                      <to variable="InputCounter"/></copy>
             </assign>
         </sequence>
    </scope>
</while>
```

Der zugehörige ForEach-Prozess hat die im folgenden Listing dargestellte Gestalt: für den *Prüfe Bestellung* Task werden in Analogie zum seriellen Fall den Pins korrespondierende Variablen spezifiziert, die `receive`-Aktivität wartet auf den Aufruf des ForEach-Prozesses und legt das Eingabeelement in die `PruefTaskInputVar` ab, die dem *Prüfe Bestellung* Task korrespondierende `invoke`-Aktivität als Eingabevariable dient und deren Ausgabe mittels einer One-way `invoke`- Aktivität an den aufrufenden Prozess zurückgesendet wird.

```
<process name="ForEachProcessPruefeBestellung" ...>
    ...
    <variables>
        <variable name="PruefTaskInputVar"   message="BestellungMsg"/>
        <variable name="PruefTaskOutputVar"  message="gpBestellungMsg"/>
    </variables>...
    ...
    <receive operation="startForEachProcess"    variable="PruefTaskInputVar" .../>
    <invoke  inputVariable="PruefTaskInputVar"  outputVariable="PruefTaskOutputVar" .../>
    <invoke  operation="CallBackForEachProcess" variable="PruefTaskOutputVar" .../>
    ...
</process>
```

Eine andere Möglichkeit der Realisierung von ForEach-Tasks besteht darin, die in der BPEL4WS Issue Liste vorgeschlagene `foreach`-Aktivität (Issue 147) [17], für die zum jetzigen Zeitpunkt noch nicht entschieden ist, ob sie in die zukünftige BPEL4WS Spezifikation mit aufgenommen wird, für die Abbildung von ForEach-Tasks zu verwenden, da sie die Fähigkeit besitzt XML Strukturen interativ, entweder mit serieller oder paralleler Semantik äquivalent zum ForEach-Task, durch eine Aktivität zu verarbeiten.

Die Syntax dieses Konstrukts ist wie folgt definiert:

```
<foreach iteratorVariableName="name" iteratorVariableType="qname" parallel="boolean"
         standard-attributes>
    <expression expressionLanguage="languageURI"> expression </expression>
    activity
</foreach>
```

Hierbei spezifiziert das `expression`-Konstrukt eine Menge von Elementen anhand eines in der durch das `expressionLanguage`-Attribut spezifzierten Sprache festgelegten Ausdrucks, von denen jeweils eins bei jeder Iteration der Variable `iteratorVariableName`, deren durch `iteratorVariableType` spezifizierten Typ mit dem Typ der einzelnen Elemente übereinstimmen muss, zugewiesen und über diese durch die von `activity` spezifizierten Aktivitäten verarbeitet wird. Anhand des `parallel`-Attributs wird zusätzlich festgelegt, ob die einzelnen Iterationen parallel (`parallel="true"`) oder seriell (`parallel="false"`) ausgeführt werden.

Damit würde sich das Problem der Abbildung von ForEach-Tasks in BPEL4WS sehr elegant, aufgrund der Verwendung der `foreach`-Aktivität, gegenüber dem im oberen Teil dieses Abschnitts dargestellten Ansatzes, lösen lassen, indem die `expression` beispielsweise einen XPath-Ausdruck spezifiziert, der alle vorhandenen Einträge der Eingabekorrelations-Variable zurückliefert, die entweder parallel oder seriell, für jedes Element durch die dem ForEach-Block korrespondierenden Aktivitäten verarbeitet werden. Zusätzlich muss eine `assign`-Aktivität spezifiziert werden, die die jeweils verarbeiteten Elemente an die durch eine Zähl-Variable spezifizierte Stelle in der Ausgabekollektions-Variable kopiert.

Die Abbildung des *Prüfe Bestellungen* ForEach-Tasks unter Verwendung der `foreach`-Aktivität zeigt das folgende Listing: die Iterator Variable ist die dem Input-Pin des Prüfe Bestellung Tasks korrespondierende `PruefTaskInputVar`-Variable. Die Menge an Elementen, die iterativ bearbeitet wird, wird anhand eines XPath-Ausdrucks `/BestellungKollektion/Bestellungen/EingabeKBestellung` charakterisiert. Die `assign`-Aktivität kopiert die Ausgabe-Daten an die durch einen Zähler `InputCounter` bestimmte Position in der Ausgabekollektions-Variable `gpBestellungKollektion`.

```
<variables>
    <variable name="InputCounter" type="int"/>
</variable>
...
<foreach iteratorVariableName="PruefTaskInputVar" iteratorVariableType="BestellungType"
         parallel="true|false">
    <expression expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116">
         /BestellungKollektion/Bestellungen/EingabeKBestellung
    </expression>
    <scope>
        <variables>
            <variable name="PruefTaskOutputVar" message="gpBestellungMsg"/>
        </variables>
        <sequence>
            <invoke inputVariable="PruefTaskInputVar" outputVariable="PruefTaskOutputVar" ../>
            <assign>
                <copy><from variable="PruefTaskOutputVar"/>
                      <to variable="gpBestellungKollektion" part="gpBestellungen"
                          query="/AusgabeKTypBestellung[InputCounter]"/></copy>
                <copy>
                      <from expression="bpws:getVariableData('InputCounter')+1"/>
                      <to variable="InputCounter"/></copy>
            </assign>
        </sequence>
    </scope>
</foreach>
```

Abschließend sei an dieser Stelle bermekt, dass der Zweite dem Ersten als möglicher Ansatz zur Abbildung von ForEach-Tasks vorzuziehen ist, da dieser zum einen die Übersichtlichkeit von BPEL4WS Prozessen besser bewahrt, da die ForEach-Tasks schneller durch korrespondierende `foreach`-Aktivitäten zu charakterisieren sind als gegenüber den `while`-Konstrukten des ersten Ansatzes, wodurch zusätzliche Änderungen in BPEL4WS leichter vom Prozessdesigner durchgeführt werden können. Zum Zweiten werden die ForEach-Blocks parallelisierter ForEach-Tasks als BPEL4WS Prozesse realisiert, was semantisch der Bedeutung eines strukturierten Tasks im BPDM wiederspricht.

Speziell für die BPEL4WS Ausführungsumgebung bedeutet dies, gegenüber dem ersten Ansatz, dass das `foreach`-Konstrukt implementiert werden muss. Dies kann beispielsweise für den seriellen Fall, zur Instanzierungszeit des Prozesses, anhand einer Substitution basierend auf dem im oberen Teil des Abschnitts dargestellten Ansatzes realisiert werden. Der parallelisierte Fall kann als solches ebenfalls mittels einer Substitution implementiert werden, was aber speziell eine zusätzliche Datenabhängigkeitsanalyse bzgl. der Variablen, die von den Aktivitäten benutzt werden und außerhalb des Scopes der `foreach`-Aktivität definiert wurden, vorraussetzt, wodurch aufgrund der Komplexität in diesem Fall eine direkte Implementierung anzustreben ist.

6.3 Subprozesse

Da BPEL4WS das Subprozess-Konzept derzeit nicht unterstützt, besteht ohne die Einführung von BPEL4WS Extensions keinerlei Möglichkeit explizit zu beschreiben, dass ein Partner-Prozess in einem Szenario als Subprozess bezüglich eines Prozesses fungiert. Es wäre zwar möglich die Subprozess-Semantik anhand der Einführung einer zusätzlichen expliziten Terminierungsoperation zu realisieren auf die jeder Prozess mit einem `eventHandler` mit dem Aufruf der `terminate` Aktivität reagiert und die vor Beendigung des Vater-Prozesses von diesem aufgerufen wird. Dies würde zur Modellierungszeit aber lediglich einen impliziten Schluss darüber zulassen, welcher Partner-Prozess speziell als Subprozess genutzt wird. Darüberhinaus würde das Lebenszyklus-Management teilweise in den Bereich der Prozessdefinition verlagert, wodurch die strikte Trennung zwischen Geschäftsprozess- und Implementierungslogik in BPEL4WS aufgehoben wird.

Daher soll im Folgenden die Einführung eines `isSubProcess`-Attributs, jeweils im Kontext des `process`- und `partnerLink`-Elements, diskutiert werden, welches explizit das Subprozess-Konzept in BPEL4WS einführt und zur Modellierungszeit einen Schluss über die jeweilige Subprozess-Relation zwischen Prozessen, als auch die funktionelle Trennung zwischen Prozessdefinition und BPEL4WS Ausführungsumgebung, ermöglicht.

6.3.1 `process`-Element

Eine Möglichkeit besteht darin das `isSubProcess`-Attribut im Kontext des `process`-Elements zu verwenden, wodurch BPEL4WS Prozesse explizit als Subprozesse identifiziert werden können. Dadurch würden prinzipiell alle BPDM Subprozesse als BPELWS Prozesse mit `isSubProcess=true` direkt abbildbar.

Rein konzeptuell besteht bei diesem Ansatz das Problem, dass durch das Subprozess-Attribut auf Prozess-Ebene für alle Instanzen des Prozesses festgelegt ist, dass diese gegenüber aufrufenden Partner-Instanzen eine Lebenszyklusabhängigkeit besitzen. Damit könnte in keiner Kollaboration der Prozess als eigenständiger Partner-Prozess verwendet werden, was im Widerspruch zur Subprozess-Definition des BPDM steht, bei dem Prozesse sowohl als Sub- als auch als Partner-Prozesse verwendet werden können.

Zusätzlich muss hierbei betrachtet werden, in wie weit die Lebenszyklusabhängigkeit des Subprozesses von seinem Vater-Prozess realisiert werden kann. Im Falle das Vater- und Subprozess von der gleichen BPEL4WS Ausführungsumgebung ausgeführt werden, reicht es, dass sich der Vater-Prozess die aufgerufenen Subprozessinstanzen merkt, die bei dessen Beendingung automatisch beendet werden. Dagegen im Falle der Ausführung von Prozess- und Subprozessinstanz durch verschiedene Ausführungsumgebungen, ergibt sich das Problem, dass die Subprozess Umgebung auf „irgendeine Art und Weise“ über die Terminierung der Vater-Prozessinstanz informiert werden muss. Die einfachste Möglichkeit, ohne zusätzliche Protokolle zwischen den Umgebungen aufzusetzen, würde beispielsweise darin bestehen, eine Restriktion für Subprozesse zu definieren, nach der alle vom Subprozess durchgeführten Aktivitäten von einer `receive`- und korrespondierenden `reply`-Aktivität umschlossen sein müssen. Dadurch würde gewährleistet, dass die Subprozessinstanz nie länger als die Vater-Prozessinstanz existiert. Diese Restriktion würde sich natürlich auf die abzubildenden BPDM Subprozesse übertragen, wodurch nicht jeder Prozess, der als Subprozess verwendet wird, abbildbar ist.

6.3.2 `partnerLink`-Element

Eine andere Möglichkeit besteht darin die Subprozess Definition auf Ebene der PartnerLink Definition einzuführen, wodurch die mit dem Prozess interagierenden Partner-Prozesse wahlweise durch die Angabe des `isSubProcess`-Attributes als Subprozesse oder reguläre Partner-Prozesse deklariert werden können. Dies hat gegenüber dem vorher diskutierten Ansatz den Vorteil, dass lediglich bezüglich des Prozesses definiert wird, welcher Partner-Prozess bei der Interaktion als Subprozesse behandelt wird. Damit können Partner-Prozesse gegenüber verschiedenen Prozessen, mit denen sie interagieren, als Subprozesse oder reguläre Partner-Prozesse verwendet werden. Dies entspricht der Semantik der im BPDM verwendeten Subprozess-Definition.

Da die Interaktion des Prozesses mit seinem Subprozess über den Aufruf von Porttyp Operationen und damit beispielsweise den Austausch von SOAP Nachrichten abgewickelt wird, bleibt an dieser Stelle die Frage, in wie weit beim initialen Aufruf übermittelt werden kann, dass die BPEL4WS Ausführungsumgebung den aufzurufenden Prozess als Subprozess instanzieren soll. Dies könnte beispielsweise anhand propritärer Protokolle zwischen den Ausführungsumgebungen oder vielleicht mittels im Web Service Umfeld vorhandenen Standard, beispielsweise WS-Coordination, realisiert werden. Darüberhinaus bleibt in Äquivalenz dazu die Frage, in wieweit die Terminierung der Vater-Prozessinstanz der Subprozessinstanz signalisiert werden kann? Dies könnte beispielsweise ebenfalls durch zusätzliche Protokolle realisiert werden.

6.4 Multiple Daten Input- und Output-Pins

Wie bereits im Abschnitt 5.2.6 dargestellt, besteht die Schwierigkeit, wie die Tasks zugeordneten mehreren Daten Input- und Output-Pins in BPEL4WS bei der Abildung von Tasks anhand von `invoke`-Aktivitäten, die lediglich über eine Eingabe- und Ausgabevariable Daten bekommen bzw. zurückliefern, in BPEL4WS in geeigneter Weise realisiert werden können.

Um den hier vorgestellten Ansatz zu verdeutlichen, wird dieser anhand des in Abbildung 6.3 dargestellten *Prüfe Liquidität* Task veranschaulicht. Dieser bekommt als Eingabe über zwei Daten-Pins die *Bestellung* sowie die *Kundendaten des Bestellers*, anhand derer geprüft wird, ob der Kunde die *Bestellung* begleichen kann. Ist dies der Fall, wird die *Bestellung* zur Weiterverarbeitung an andere Tasks weitergeleitet. Ist dies nicht der Fall, wird ein *Ablehnungsbescheid* zurückgeliefert.

Ein möglicher Ansatz die über mehrere Daten Input- und Output-Pins dem Task zu- bzw. abgeführten Daten der korrespondierenden BPEL4WS `invoke`-Aktivität zu- bzw. abzuführen, besteht darin, alle korrespondierenden Daten der Daten-Pins in jeweils der Ein- bzw. Ausgabevariable zu subsum-

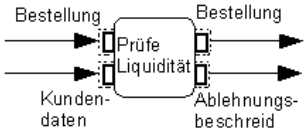

Abbildung 6.3: *Beispiel Tasks Prüfe Liquidität.*

mieren, die somit an die `invoke`-Aktivität übergeben bzw. von ihr abgeführt werden können. Der jeweilige durch die `invoke`-Aktivität auszuführende Web Service wird hierbei so realisiert, dass er aufgrund der eingehenden Daten jeweils das dem Task spezifische Verhalten realisiert und entsprechende Rückgabewerte produziert. Aufgrund der in Abschnitt 5.1.2 beschriebenen Art und Weise der Abbildung von Daten Pins, bei der jeder Pin anhand einer Variablen, deren Typ durch eine WSDL-Message spezifiziert ist, charakterisiert wird, kann dies wie folgt realisiert werden: Es wird für alle Daten Input- bzw. Output-Pins gleichzeitig neben den aus der Abbildung resultierenden Variablen eine Variable erzeugt, die anhand einer WSDL-Message, die alle `parts` der WSDL Messages der Einzel-Variablen subsummiert, getypt ist, die als Ein- bzw. Ausgabevariable für die dem Task korrespondierenden `invoke`-Aktivität verwendet werden und die vor bzw. nach dieser im BPEL4WS Dokument jeweils anhand einer `assign`-Aktivität, falls es sich um die Eingabevariable handelt, mit den Werten der den Input-Pins korrespondierenden Variablen initialisiert wird bzw. falls es sich um die Ausgabevariable handelt, die Output-Pins korrespondierenden Variablen mit den Rückgabewerten der `invoke`-Aktivität belegt. Für den Fall, dass mehrere `parts` unterschiedlicher WSDL Message den selben Namen besitzen, werden diese anhand eines Alias, beispielsweise dem `part`-Namen mit einem zusätzlichen Präfix, in der die einzelnen `parts` enthaltenden WSDL Message unterschieden.

Das folgende Beispiel soll diesen Ansatz anhand des im oberen Absatz dargestellten Beispiels schematisch demonstrieren. Die Typen der Daten Input- und Output-Pins werden anhand von WSDL Message repräsentiert. D.h. der Typ *Bestellung* als `BestellungsMsg`, die aus `partBestellung1` bis `partBestellungi` Teilen besteht, der *Kundendaten*-Typ als `KundendatenMsg`, die aus `partKundendaten1` bis `partKundendatenj` Teilen besteht sowie der Typ *Ablehnungsbescheid*, als `AblehnungsbescheidMsg`, die aus `partAblehnungsbescheid1` bis `partAblehnungsbescheidl` Teilen besteht, dargestellt im folgenden Listing.

```
<message name="BestellungMsg">
    <part name="partBestellung1" type="partBestellung1Typ"/>
    ...
    <part name="partBestellungi" type="partBestellungiTyp"/>
</message>

<message name="KundendatenMsg">
    <part name="partKundendaten1" type="partKundendaten1Typ"/>
    ...
    <part name="partKundendatenj" type="partKundendatenjTyp"/>
</message>

<message name="AblehnungsbescheidMsg">
    <part name="partAblehnungsbescheid1" type="Ablehnungsbescheid1Typ"/>
    ...
    <part name="partAblehnungsbescheidl" type="AblehnungsbescheidlTyp"/>
</message>
```

Zusätzlich werden zwei weitere WSDL-Messages erzeugt: `LiquiditaetPInputMsg`, die alle `parts` der WSDL-Messages der korrespondierenden Input-Pins (`BestellungMsg`, `KundendatenMsg`) und `LiquiditaetPOutputMsg`, die alle `parts` der WSDL-Messages der korrespondierenden Output-Pins (`Be-`

stellungMsg, AblehnungsbescheidMsg) enthält; schematisch dargestellt anhand des nachfolgenden Listings:

```
<message name="LiquiditaetPInputMsg">
    <part name="partBestellung1" type="partBestellung1Typ"/>
    ...
    <part name="partBestellungi" type="partBestellungiTyp"/>
    <part name="partKundendaten1" type="partKundendaten1Typ"/>
    ...
    <part name="partKundendatenj" type="partKundendatenjTyp"/>
</message>

<message name="LiquiditaetPOutputMsg">
    <part name="partBestellung1" type="partBestellung1Typ"/>
    ...
    <part name="partBestellungi" type="partBestellungiTyp"/>
    <part name="partAblehnungsbescheid1" type="Ablehnungsbescheid1Typ"/>
    ...
    <part name="partAblehnungsbescheidl" type="AblehnungsbescheidlTyp"/>
</message>
```

Diese werden zur Typisierung der für die den *Prüfe Liquidität* Task korrespondierenden invoke-Aktivität vorgesehenen Ein- (LiquiditaetPTaskInputVar) und Ausgabevariable (LiquiditaetP-TaskOutputVar) verwendet. Zusätzlich werden den Daten-Pins korrespondierende BPEL4WS Variablen (Bestellung, Kundendaten, Ablehnungsbescheid) erzeugt, die jeweils anhand BestellungMsg, KundendatenMsg und AblehnungsbescheidMsg typisiert sind. Das folgende Listing zeigt die schematische Definition dieser Variablen in BPEL4WS.

```
<variables>
    <variable name="LiquiditaetPTaskInputVar"  message="LiquiditaetPInputMsg"/>
    <variable name="LiquiditaetPTaskOutputVar" message="LiquiditaetPOutputMsg"/>

    <variables name="Bestellung"         message="BestellungMsg"/>
    <variables name="Kundendaten"        message="KundendatenMsg"/>
    <variables name="Ablehnungsbescheid" message="AblehnungsbescheidMsg"/>
</variables>
```

Der *Prüfe Liquidität* Task wird anhand einer invoke-Aktivität repräsentiert, der als Eingabevariable LiquiditaetPTaskInputVar und Ausgabevariable LiquiditaetPTaskOutputVar verwendet. Da sich die jeweiligen Eingabedaten in den, den Daten Input Pins korrespondieren Variablen befinden (siehe hierzu 5.1.2, Abbildung von Datenkanten, *Flow (isControl=false)*), wird die Eingabevariable der invoke-Aktivität mit deren Werten anhand einer assign-Aktivität, die alle parts der Eingabevariablen mit den Werten der korrespondierenden parts der Input-Pin Variablen initialisiert, belegt. Eine sich an die invoke-Aktivität anschließende assign-Aktivität sorgt für die Belegung der den Output-Pins korrespondierenden Variable anhand der Werte der einzelnen parts der LiquiditaetPTaskOutputVar.

```
<assign>
    <copy><from variable="Bestellung" part="partBestellung1"/>
          <to variable="LiquiditaetPTaskInputVar" part="partBestellung1"/></copy>
    ...
    <copy><from variable="Bestellung" part="partBestellungi"/>
          <to variable="LiquiditaetPTaskInputVar" part="partBestellungi"/></copy>
    ...
    <copy><from variable="Kundendaten" part="partKundendaten1"/>
          <to variable="LiquiditaetPTaskInputVar" part="partKundendaten1"/></copy>
```

```
    ...
    <copy><from variable="Kundendaten" part="partKundendatenj"/>
          <to variable="LiquiditaetPTaskInputVar" part="partKundendatenj"/></copy>
</assign>
<invoke ... inputVariable="LiquiditaetPTaskInputVar"
            outputVariable="LiquiditaetPTaskOutputVar"/>
<assign>
    <copy><from variable="LiquiditaetPTaskOutputVar" part="partBestellung1"/>
          <to variable="Bestellung" part="partBestellung1"/></copy>
    ...
    <copy><from variable="LiquiditaetPTaskOutputVar" part="partBestellungi"/>
          <to variable="Bestellung" part="partBestellungi"/></copy>

    <copy><from variable="LiquiditaetPTaskOutputVar" part="partAblehnungsbescheid1"/>
          <to variable="Ablehnungsbescheid" part="partAblehnungsbescheid1"/></copy>
    ...
    <copy><from variable="LiquiditaetPTaskOutputVar" part="partAblehnungsbescheidl"/>
          <to variable="Ablehnungsbescheid" part="partAblehnungsbescheidl"/></copy>
</assign>
```

Abschließend sei bemerkt, dass dieser Ansatz keine zusätzlichen Informationen über den durch den Task realisierten Kontrollfluss benötigt, sondern lediglich auf Basis der Daten Input- und Output-Pins, unabhängig einer Gruppierung, arbeitet, wodurch Kontroll- und Datenfluss bei der Abbildung von BPDM-Prozessen in BPEL4WS unabhängig sind und die Abbildung beispielsweise in mehreren Phasen, beispielsweise einer Kontrollfluss- und einer anschließenden Datenphase durchgeführt werden kann.

6.5 Kollaborative Prozesse

Wie bereits im Abschnitt 5.2.2 dargestellt, besteht in BPEL4WS keine Möglichkeit Geschäftsprotokolle aus globaler Sicht zu spezifizieren. Lediglich anhand abstrakter Prozesse können diese jeweils lokal, bezüglich den in der Kollaboration involvierten Rollen, charakterisiert werden. Daher besteht das Problem, dass die im BPDM mittels kollaborativen Prozessen spezifizierten Geschäftsprotokolle nicht in dieser Form in BPEL4WS abgebildet werden können.

Da kollaborative und abstrakte Prozesse äquivalent ineinander abbildbar sind (siehe hierzu Abschnitt 4.2.1), besteht ein möglicher Ansatz darin, die durch kollaborative Prozesse spezifizierten Geschäftsprotokolle durch abstrakte BPDM Prozesse zu repräsentieren, die in BPEL4WS als abstrakte Prozesse abgebildet werden und somit das Geschäftsprotokoll der Kollaboration, zwar nicht aus globaler Sicht, in BPEL4WS beschreiben.

Hierzu bedarf es der Spezifikation der Abbildung von kollaborativen in abstrakte Prozesse, die bisher nicht weiter charakterisiert wurde, für die im Folgenden ein möglicher Ansatz skizziert werden soll. Da kollaborative Prozesse zur Zeit nur sehr unvollständig charakterisiert sind und weder Informationen über deren Elemente und Semantik vorhanden sind, werden hierfür zu Beginn die wesentlichen Elemente zur Beschreibung von kollaborativen Prozessen definiert, anhand derer die Abbildung spezifiziert wird. Hierbei wird versucht die Spezifikation so allgemein wie möglich zu halten, so dass der hier vorgestellte Ansatz zu einem späteren Zeitpunkt auf BPDM kollaborative Prozesse übertragen werden kann.

Ein kollaborativer Prozess besteht aus folgenden Elementen: Knoten, die jeweils eine Phase der Interaktion zwischen zwei Rollen, anhand eines vergleichbaren UML Sequenzdiagramms, aus dem der

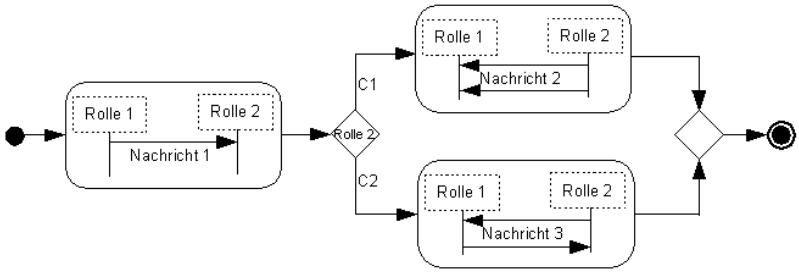

Abbildung 6.4: *Schema eines kollaborativen Prozesses.*

zeitliche Ablauf des Nachrichtenaustausch ersichtlich wird, im Protokoll spezifizieren, gerichteten Kanten, die Knoten verbinden und die den Kontrollfluss des kollaborativen Prozesses beschreiben, Verzweigungsknoten, die bezüglich einer Rolle spezifiziert sind, anhand derer verschiedene alternative Pfade im Protokoll, mittels einer empfangenen Nachricht von dieser Rolle, beschrieben werden, die von Verschmelzungsknoten zusammengeführt werden können, sowie Kontrollfluss-Start- als auch Kontrollfluss-Ende Knoten. Abbildung 6.4 zeigt ein Schema eines solchen kollaborativen Prozesses. Hierbei sind Interaktionsknoten als Rechtecke mit abgerundeten Ecken symbolisiert, Verzweigungs- und Verschmelzungsknoten als Diamanten-Symbole wobei Verzweigungsknoten zusätzlich mit dem Name der Rolle und den jeweiligen Bedingungen, an den Kanten, notiert sind und Kontrollfluss-Start- als ausgefüllte Kreise und -Ende Knoten als ausgefüllte Kreise mit zusätzlich umrandenden Kreis erkennbar.

Zusätzlich sollen für die hier charakterisierten kollaborativen Prozesse, Zwecks Erhaltung der Konsistenz zu abstrakten Prozessen und zur vereinfachten Darstellung dieses Ansatzes, die folgenden Eigenschaften gelten: der Kontrollfluss von kollaborativen Prozessen ist azyklisch, die durch die direkt dem Verzweigungsknoten nachfolgenden Interaktions-Knoten beschriebenen Interaktionen müssen mit dem Empfang bzw. dem Versenden einer Nachricht bzgl. der für die den Verzweigungsknoten spezifizierten Rolle eingeleitet werden. Zusätzlich müssen alle versendeten Nachrichten unterschiedlich sein und die Empfängerrolle in jeweils einem anderen Pfad ebenfalls eine Nachricht empfangen, wenn die Empfängerrollen in den beteiligten Pfaden sich unterscheiden. Zusätzlich gilt, dass Rollen in einem Pfad auch in den jeweiligen anderen Pfad auftreten müssen.

Unter diesen Annahmen können die hier spezifizierten kollaborativen Prozesse, die eine Kollaboration zwischen den Rollen R_i $(i = 1, \ldots, n)$, $n \in \mathbb{N}$ beschreiben, als abstrakte Prozesse für jeweils eine Rolle R_i wie folgt abgebildet werden:

1. Überführe den kollaborativen Prozess für jede Rolle R_i in einen reduzierten „kollaborativen“ Prozess, indem
   (a) Interaktionsknoten, die die Rolle R_i nicht enthalten, übergangen werden.
   (b) Interaktionsknoten, bei denen die Rolle R_i beteiligt ist, als Sequenz von Sende- und Empfangsknoten bzgl. dieser Rolle wie folgt abgebildet werden: Zu jeder Nachricht, die die Rolle R_i sendet, wird ein Sendeknoten $S_{T,M}$ erzeugt, wobei T die Zielrolle und M den Namen der Nachricht bezeichnet, auf jede Nachricht, auf die die Rolle wartet, ein Empfangsknoten $E_{S,M}$, wobei S die Quell-Rolle bezeichnet. Sende- und Empfangsknoten werden anhand von gerichteten Kanten verbunden wobei gilt, dass die Reihenfolge von Sende- und Empfangsknoten im Kontrollfluss der zeitlichen Reihenfolge des Auftretens der Aktivitäten im Sequenzdiagramm bzgl. Rolle R_i entspricht.
   (c) Kontrollfluss-Start- und End-Knoten erhalten bleiben.

(d) Verzweigungs- und Verschmelzungsknoten sowie deren Bedingungen übernommen werden.

(e) Kontrollflusskanten ebenfalls beibehalten werden.

2. Bilde die reduzierten „kollaborativen“ Prozesse für jede Rolle R_i in abstrakte BPDM Prozesse ab, indem:

   (a) Kontrollfluss-Start- und End-Knoten als BPDM Start- und End-Knoten repräsentiert werden.

   (b) Empfangsknoten $R_{S,M}$ in ReceiveMessageTasks bzgl. der Nachricht M und dem Sender S abgebildet werden.

   (c) Sendeknoten $S_{T,M}$ in SendMessageTasks bzgl. der Nachricht T und dem Empfänger T abgebildet werden.

   (d) Verzweigungsknoten für die Rolle R_i als ein Task repräsentiert werden, der mehrere alternative Kontroll-Output-Pins (eins für jeden Pfad) besitzt, so dass alle weiteren Knoten in den dafür äquivalenten BPDM Pfad abgebildet werden können.

   (e) Verzweigungsknoten für die Rolle R_j mit $i \neq j$, deren nachfolgenden Knoten alles Empfangsknoten sind, als ReceiveMessageTask abgebildet werden, die auf alle diese Nachrichten reagieren können und entsprechend die gleiche Anzahl an Kontroll-Output-Pins besitzen. Alle weiteren Knoten werden in den dafür äquivalenten BPDM Pfad abgebildet.

   (f) Verschmelzungsknoten, die einen nachfolgenden Empfangs- bzw. Sendeknoten besitzen, in einen Receive- bzw. SendMessageTask mit der gleichen Anzahl an Kontroll-Input-Pins wie eingehende Kanten des Verschmelzungsknoten abgebildet werden.

   (g) Verschmelzungsknoten mit einem Verschmelzungsknoten oder End-Knoten als Nachfolger in Form eines (empty) Tasks, der über die gleiche Anzahl an Kontroll-Input-Pins wie eingehende Kanten verfügt, abgebildet werden.

   (h) die Kanten zwischen den Knoten in äquivalente BPDM Kontrollflusskanten überführt werden.

Abbildung 6.5: *Bestell-Kollaboration zwischen den Rollen: Käufer, Verkäufer und Lieferant.*

Zur Verdeutlichung der Abbildung kollaborativer in abstrakte Prozesse wird diese im Folgenden anhand des in Abbildung 6.6 dargestellten kollaborativen Prozesses bezüglich der *Bestell-Kollaboration* (Abbildung 6.5) demonstriert. Die *Bestell-Kollaboration* beschreibt eine Kollaboration zwischen drei Rollen: *Käufer*, *Verkäufer* und *Lieferant*, deren Ablauf anhand des kollaborativen Prozesses ersichtlicht ist: Zu Beginn sendet der *Käufer* dem *Verkäufer* seine *Bestellung*. Dieser leitet die *Bestellung* an einen *Lieferanten* weiter, der überprüft, ob die *Waren* lieferbar sind. Der daraus resultierende *Lieferstatus* wird an den *Verkäufer* zurückgesendet, der aufgrund dieses *Lieferstatus* entweder dem *Käufer* eine *Fehlermeldung* „die Ware ist nicht lieferbar„ bzw. eine *Versandbestätigung*, die ihn über den erfolgreichen Versand der Ware informiert, sendet.

Im ersten Schritt der Abbildung werden bezüglich *Käufer*, *Verkäufer* und *Lieferant* reduzierte „kollaborative“ Prozesse erzeugt, die in Abbildung 6.7 dargestellt sind.

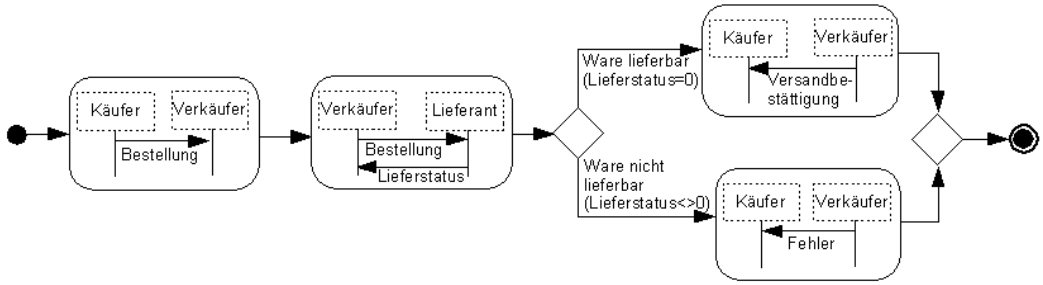

Abbildung 6.6: *Kollaborativer Prozess zur Bestell-Kollaboration.*

Anschließend werden diese reduzierten „kollaborativen“ Prozesse, die jeweils das Verhalten einer Rolle in der Kollaboration beschreiben, in abstrakte BPDM Prozesse nach den im vorherigen Absatz erwähnten Punkten abgebildet, so dass die im Abbildung 6.8 dargestellten abstrakten BPDM Prozesse erzeugt werden. Diese werden abschließend in äquivalente BPEL4WS abstrakte Prozesse transformiert, die die Kollaboration in BPEL4WS charakterisieren. Hierfür werden zusätzlich, bezogen auf die Kollaboration, PartnerLink Typen, Partner Links und Variablen erzeugt, anhand derer die jeweiligen Partner und Rollen spezifiziert werden und die zur Aufnahme der jeweiligen empfangenen bzw. zu sendenden Nachrichten bezüglich Receive- und SendMessageTasks dienen.

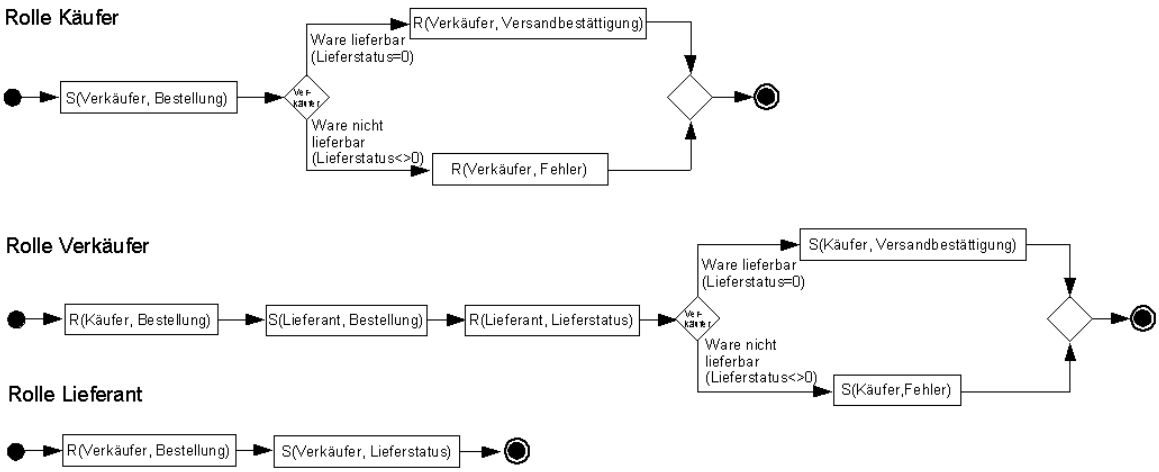

Abbildung 6.7: *Reduzierte kollaborative Prozesse für Käufer, Verkäufer und Lieferant.*

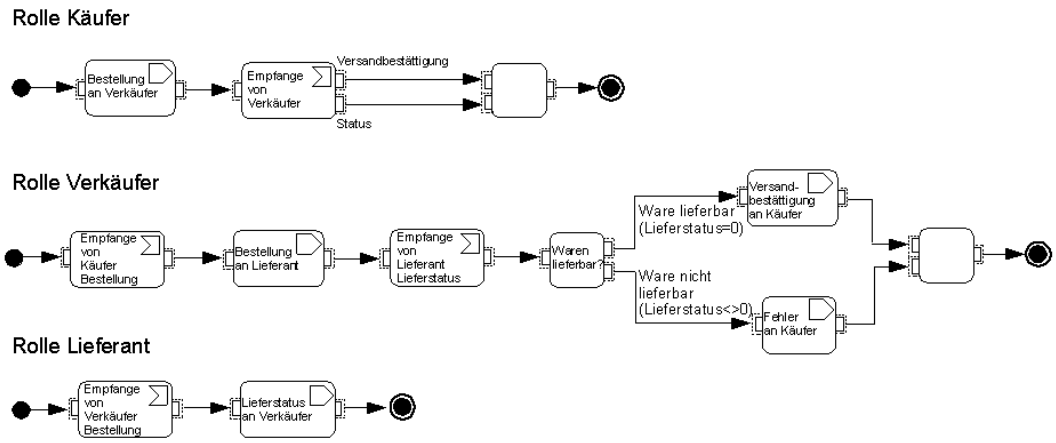

Abbildung 6.8: *Abstrakte BPDM Prozesse für Käufer, Verkäufer und Lieferant.*

Für das hier verwendete Beispiel sieht der abstrakte BPEL4WS Prozess für die *Verkäufer* Rolle wie folgt aus:

```
<process isAbstract="true" ...>
    <partnerLink name="KaeuferVerkaeuferPL"
                 partnerLinkTyp="KaeuferVerkaeuferPLT"
                 myRole="Kaeufer"/>
    <partnerLink name="Verk"auferLieferantPL"
                 partnerLinkTyp="VerkaeuferLieferantPLT"
                 myRole="Kaeufer"/>

    <variables>
        <variable name="Bestellung" .../>
        <variable name="Lieferstatus" .../>
    </variables>
    ...
    <sequence>
       <receive partner="KaeuferVerkaeuferPL"   operation="getBestellung"
                variable="Bestellung" .../>
       <invoke  partner="VerkaeuferLieferantPL" operation="sendBestellung"
                inputVariable="Bestellung" .../>
       <receive partner="VerkaeuferLieferantPL" operation="getLieferstatus"
                variable="Lieferstatus" .../>
       <switch>
           <case condition="Lieferstatus=0">
               <invoke partner="KaeuferVerkaeufer" operation="sendVersandbestaettigung".../>
           </case>
           <case condition="Lieferstatus<>0">
               <invoke partner="KeauferVerkeaufer" operation="sendFehler".../>
           </case>
        </switch>
        <empty/>
    </sequence>
</process>
```

6.6 Repository

Repositories stellen im BPDM Modellelemente dar, mit denen Daten in Form von Token über die Dauer des Lebenszyklus eines Prozesses persistent gehalten sowie zwischen mehreren Prozessen ausgetauscht werden können. In BPEL4WS existiert ein deratiges Konzept nicht, da lediglich Variablen, die Daten lokal bezüglich der jeweiligen Prozessinstanz halten, die nach deren Beendigung verloren gehen, zur Datenhaltung einsetzbar sind. Damit ergibt sich, wie bereits in Abschnitt 5.2.7 dargestellt, die Schwierigkeit Repositories in BPEL4WS in geeigneter Weise abzubilden.

Da Repositories sehr große Ähnlichkeit mit Datenbanken besitzen, auf die innerhalb verschiedener BPDM Prozesse zugegriffen werden kann, besteht ein möglicher Ansatz zur Abbildung von Repositories in BPEL4WS darin, diese anhand von Datenbanken, deren Datensätze den von Repositories enthaltenen Token entsprechen, zu realisieren, auf die über Web Services, in analoger Weise zu Repositories im BPDM, aus BPEL4WS Prozessen zugegriffen werden kann. Dadurch können die Daten von Prozessen über deren Lebenszyklus persistent gehalten als auch zwischen verschiedenen BPEL4WS Prozessinstanzen ausgetauscht werden.

Für diesen Zugriff reichen im wesentlichen zwei Operationen, die vom Web Service angeboten werden, aus: eine Operation, mit der Daten in die Datenbank mit Überschreibungssemantik eingebracht werden können (zum Beispiel `write(Object)`) und die im BPEL4WS Prozess an den Stellen aufgerufen wird, bei denen im korrespondierenden BPDM Prozess Token in Repositories emittiert werden und eine Operation, mit der Daten aus der Datenbank abgerufen werden können (zum Beispiel `Object read(modus)`), die jeweils an den Stellen im BPEL4WS Prozess aufgerufen wird, bei denen im korrespondierenden BPDM Prozess ein Task zusätzlich Token aus einem Repository konsumiert. Die Operation zum Abrufen von Daten muss zusätzlich zwei Zugriffmodi, in Äquivalenz zur Copy- und Remove-Semantik im BPDM, unterstützen: den Copy-Modus, der den abzurufenden Datensatz in der Datenbank beläßt und den Remove-Modus, der diesen löscht. Darüber hinaus müssen alle Datensätze innerhalb der Datenbank so linearisierbar sein, dass speziell das durch die vom Repository verwendete Einfügestrategie bestimmte Stack- bzw. Warteschlangenverhalten beim Abrufen von Daten realisiert werden kann.

Die Abbildung gestaltet sich demnach so, dass alle in das Repository eingehenden und ausgehenden Kanten an der jeweiligen Stelle im Kontrollfluss durch den Aufruf der `write`-Operation bzw. der `read`-Operation mit Hilfe einer `invoke`-Aktivität in BPEL4WS charakterisiert werden, die, falls beim Schreiben die Kapazitätsgrenze erreicht ist, eine Fehlermeldung zurückliefert.

Das der Datenbank zugrundeliegende Datenbank-Schema kann hierbei beispielsweise anhand aus der durch eine Item Definition typisierte Repository korrespondierenden WSDL Message und zugehörigen XML Schemas erzeugt werden, indem beispielsweise bei der Verwendung von relationalen Datenbanken jeder Part der WSDL Message als Tabelle realisiert wird, deren Struktur aus dem zugehörige XML Schema abgeleitet wird.

Die praktische Umsetzung dieses Ansatzes kann auf unterschiedliche Art und Weisen erfolgen. Beispielsweise können als Datenbanken relationale, objekt-relationale, objekt-orientierte oder sogar XML-basierte Datenbanken zum Einsatz kommen, auf die entweder direkt, durch die von ihnen zur Verfügung gestellten Web Service Schnittstellen, oder über einen speziell auf die Eigenschaften von Repositories zugeschnittenen Web Service von BPEL4WS aus zugegriffen werden kann.

Speziell für den Prozess der Abbildung von BPDM Prozessen in BPEL4WS bedeutet dieser Ansatz, dass zusätzliche Informationen für die Abbildung von Repositories durch den Anwender spezifiziert werden müssen, die sich beispielsweise auf das zu verwendete Datenbank Management System (DBMS)

und die jeweiligen Web Services beziehen.

6.7 manuelle Tasks, Ressourcen

Wie bereits in Abschnitt 5.2.9 dargestellt, ist BPEL4WS als Sprache zur Beschreibung ausführbarer Geschäftsprozesse, bei denen Arbeitsschritte durch Web Services realisiert werden, konzipiert, die keinerlei Unterstützung zur Spezifikation von Personen durchzuführenden Arbeitsschritten im Gegensatz zum BPDM bietet. Dadurch besteht das Problem, wie manuelle BPDM Tasks und damit verbunden die jeweiligen Ressourcenbeschreibungen in BPEL4WS geeignet abgebildet werden können.

Da es grundsätzlich außerhalb des Kontext von BPEL4WS liegt manuelle Arbeitsschritte zu spezifizieren, sei an dieser Stelle bemerkt, dass es von diesem Standpunkt aus wenig Sinn macht manuelle BPDM Arbeitsschritte in BPEL4WS abzubilden. Trotz dieser Tatsache soll in diesem Abschnitt ein Ansatz dargestellt werden, mit dem auf Basis von BPELWS Erweiterungen, die hier lediglich zur Problemlösung spezifiziert sind, manuelle BPDM Tasks sowie die damit verbundenen Ressourcenmodelle in BPEL4WS abgebildet werden können.

Hierfür werden die von IBM im Kontext der WebSphere-Produktpalette gemachten Erweiterungen von BPEL4WS [24] (im Folgenden als BPEL4WS+ [24] bezeichnet) verwendet, die mit Hinblick auf die Spezifikation manueller Arbeitsschritte nachfolgend kurz charakterisiert werden und anhand derer im Anschluss daran die Abbildung manueller Tasks dargestellt wird.

In BPELWS+ werden zur Beschreibung manueller Arbeitsschritte sog. Staff-Aktivitäten verwendet, die syntaktisch als spezielle `invoke`-Aktivitäten charakterisiert sind und eine durch eine Menge von Personen ausführbare Aktivität spezifizieren. Die Ausführung von Staff-Aktivitäten gestaltet sich, indem an jede durch die Aktivität spezifizierten Personen ein Work-Item, welches die jeweils durchzuführende Aktion mit ihren Daten beschreibt, versendet wird, dass von einer dieser Personen, die sich für dessen Bearbeitung entscheidet, durchgeführt wird, indem sie die Eingabedaten ließt, die notwendigen Aktionen zur Realisierung der Aktivität ausführt und die Ausgabedaten als Ausgabenachricht der Aktivität erzeugt, die daraufhin beendet und die Ausführung des Geschäftsprozesses fortgesetzt wird.

Die Syntax von Staff-Aktivitäten ist wie folgt spezifiziert:

```
<invoke partnerLink="null" portType="QName" operation="ncName"
        inputVariable="ncname" outputVariable="ncname">

    <wpc:staff>
        <wpc:potentialOwner>staffQueryVerb</wpc:potentialOwner>
        <wpc:editor>staffQueryVerb</wpc:editor>?
        <wpc:reader>staffQueryVerb</wpc:reader>?
     </wpc:staff>
     ...
</invoke>
```

`portTyp` und `operation` beschreiben die von einer Person durchzuführende Aktion, Ein- und Ausgabevariablen charakterisieren die für die Aktion notwendigen Daten bzw. liefern die durch die Aktion produzierten Daten zurück. Das `staff`-Element spezifiziert, anhand von Staff Querys (`staffQueryVerb`), die zur Ausführung der Aktivität in Frage kommenden Personen. Staff Querys stellen hierbei Anfragen gegenüber einem Organisations-Verzeichnis (z.B. ein LDAP-Verzeichnis) dar, die eine Liste von Personen zurückliefern, den Rollen `potentialOwner`, `editor`, `reader` zugeordnet werden. `potentialOwner` charakterisiert die Personen, die Staff-Aktivitäten ausführen dürfen, `editor`, die lediglich die Daten der Aktivität sehen und verändern dürfen und `reader`, die nur die Daten der Aktivität sehen dürfen. Die Angabe der `editor`- und `reader`-Rollen sind optional.

Staff-Querys werden unabhängig eines spezifischen Organisations-Verzeichnisses in Form sog. Staff Verbs spezifiziert, die zum Deployment-Zeitpunkt der Prozesse in „reale“ Querys bezüglich des zu verwendeten Organisations-Verzeichnisses übersetzt werden um die Details und etwaige Änderungen in der Struktur der Verzeichnisse von der Prozess-Spezifikation zu trennen. Staff Verbs sind vergleichbar mit Methoden, die einen bestimmten Query-Typ charakterisieren, die über eine Menge von Parametern verfügen und anhand eines Namens identifiziert werden. Sie werden zur Design-Zeit innerhalb einer separaten XML Datei definiert, für die zusätzlich eine Transformation mittels XLST in die jeweilige vom Zielsystem verwendete Querysprache spezifiziert ist. Eine genauere Charakterisierung von Staff-Verbs sowie deren Transformation ist in [31] zu finden. Das nachfolgende Listing zeigt als Beispiel die Definition eines Staff-Verb „Finde Person nach Benutzer-ID“, welches eine Query charakterisiert, die eine Person anhand ihrer Benutzer-ID ermittelt.

```
<DefineVerb name="Finde Person nach BenutzerID"/>
    <Mandatory>
        <Parameter>
            <Name>UserID</Name>
            <Type>xsd:string</Name>
        </Parameter>
    </Mandatory>
</DefineVerb>
```

Die Verwendung des definierten Staff Verbs im Prozess zeigt das nachfolgende Listing. Die einzelnen Parameter sind mit Werten spezifiziert, anhand derer die Query exakt spezifiziert ist.

```
<staff:verb>
    <staff:name>Finde Benutzer nach ihrer BenutzerID</staff:name>
    <staff:id>User by userID</staff:id>
    <staff:parameter id="BenutzerID">100</staff:paramater>
</staff:verb>
```

In diesem Fall wird die Person als Ergebnis der Query zurückgeliefert, deren Benutzer-ID 100 ist.

Unter Verwendung der hier vorgestellten Erweiterungen von BPEL4WS lassen sich manuelle BPDM Tasks und damit verbunden die jeweiligen Ressourcenbeschreibungen in BPEL4WS+ wie folgt abbilden: manuelle Tasks werden als Staff-Aktivitäten abgebildet, deren durchzuführende Aktion anhand von `portTyp` und `operation` spezifiziert sind. Die jeweils dem manuellen Task zugeordneten Input- und Output-Pins werden nach dem in Abschnitt 6.4 dargestellten Ansatz realisiert. Die für Task spezifizierten ResourceRequirements werden als Staff Verbs charakterisiert, wobei lediglich IndividualResourceRequirements, die sich auf einen IndividualResourceTyp mit isHuman=true beziehen, und RequiredRole abbildbar sind. Die im Kontext von IndividualResourceRequirements spezifizierten IndividualResourceTyps charakterisieren das vom Organisations-Verzeichnis verwendete Datenschema und die einzelnen Ausprägungen in Form von IndividualResource (isHuman=true) die Einträge im Organisations-Verzeichnis. Das zu verwendente Organisations-Verzeichnis muss bei der Abbildung zusätzlich manuell spezifiziert werden.

Die Abbildung soll anhand des in Abbildung 6.9 dargestellten Beispiels eines manuellen *Prüfe Bestellung* Tasks skizziert werden. Dieser bekommt als Eingabe eine *Bestellung* und liefert einen *Prüfstatus* als Ausgabe zurück. Die bezüglich des Tasks definierte IndividualResourceRequirement *Angestellter-Requirement* (dargestellt als Rechteck, das mit einer gestrichelten Linie dem Task zugeordnet ist) spezifiziert als notwendige Ressource für die Ausführung des Task einen Angestellten, der aus der Abteilung *Bestellprüfung* ist. Zusätzlich sind Ressourcen anhand IndividualResource (*HansMüller*, *ErikHeinrich*) im Modell spezifiziert, welche Ausprägungen des IndividualResourceTyp *Angestellter*, der in Form einer UML Klasse dargestellt ist, sind.

Der *Prüfe Bestellung* Task wird in eine Staff-Aktivität abgebildet, dargestellt im nachfolgenden Listing, mit `BestellungVar` als Eingabe- und `PruefstatusVar` als Ausgabe-Variable. Die IndividualResourceRequirment *AngestellterRequirement* wird als Staff-Verb abgebildet, welches als potentiellen

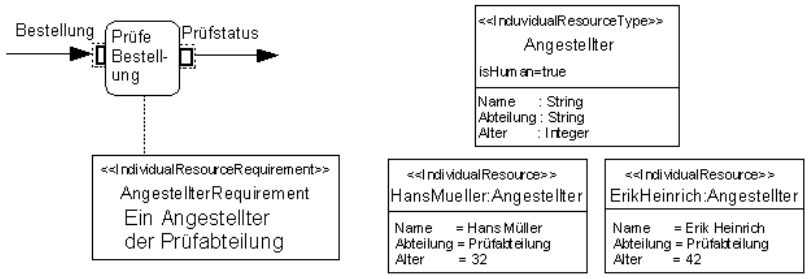

Abbildung 6.9: *Manueller Task Prüfe Bestellung mit zugehörigem ResourceRequirement und entsprechender Ressourcen-Spezifikation.*

Besitzer der Aktivität einen *Angestellten* der *Bestellprüfung*-Abteilung spezifiziert. Der Ressourcetyp wird als Daten-Schema und die jeweiligen Ressourcen als Einträge, dargestellt in Tabelle 6.1, bei der die Spalten das Schema charakterisieren und die Zeilen die einzelnen Ausprägungen, in einem Organisations-Verzeichnis abgebildet.

```
<invoke partnerlink="null" portTyp="Bestellung" operation="PruefeBestellung"
        inputVariable="BestellungVar" outputVariable="PruefstatusVar">

    <staff:verb>
<staff:potentialOwner>
            <staff:name>AngestellterQueryAbteilung</staff:name>
            <staff:parameter id="Abteilung">Pr"ufabteilung</staff:paramater>
        </staff:potentialOwner>
    </staff:verb>
</invoke>
```

ID:Integer	Name:String	Abteilung:String	Alter:Integer
1	Hans Müller	Bestellprüfung	32
2	Erik Heinrich	Bestellprüfung	42

Tabelle 6.1: *Daten-Schema sowie Daten-Einträge korrespondierend zum Ressourcenmodell dargestellt in tabellarischer Form.*

Abschließend sei an dieser Stelle bermerkt, dass mit Hilfe des hier vorgestellten Ansatzes zwar BPDM Prozesse, in denen manuelle Task vorkommen, in BPEL4WS abgebildet werden können, die daraus resultierenden BPEL4WS Prozesse aber nicht mehr portabel, indem sie in verschiedenen Ausführungsumgebung ausgeführt werden können, sind, da diese die jeweiligen BPEL4WS Erweiterungen nicht notwendiger Weise unterstützen müssen.

6.8 Korrelationsprädikate

Wie in Abschnitt 5.2.4 dargelegt, existieren bei der Abbildung von Korrelationsprädikaten auf Korrelationsmengen Probleme, die aufgrund der restriktiven Struktur von Korrelationsmengen gegenüber Korrelationsprädikaten im BPDM verursacht werden. Da diese Restriktionen zu gravierend und nicht mit den von BPEL4WS zur Verfügung gestellten Konzepten zur Prozessbeschreibung in geeigneter Form kompensiert werden können, besteht keine Möglichkeit diese Probleme ohne Erweiterung von BPEL4WS zu lösen. Da jede mögliche Erweiterung von BPEL4WS bezüglich der Realisierung der Abbildung von BPDM in BPEL4WS die Portabilität von BPEL4WS Prozessen einschränkt und speziell zur Realisierung der kompletten Semantik von Korrelationsprädikaten in BPEL4WS umpfangreiche Erweiterungen notwendig sind stellt, die Erweiterung von BPEL4WS keinen sinnvollen Ansatz für die Abbildung von Korrelationsprädikaten dar. Daher wird im Folgenden der Ansatz verfolgt, wenigstens die in BPEL4WS abbildbaren Korrelationsprädikate, als auch deren Abbildung zu charakterisieren, so dass die auftretenden Problemfälle bei der Abbildung erkannt und dem Prozess-Designer signalisiert werden können, die er beispielsweise durch eine Prozess-Restrukturierung kompensieren kann. Hierfür werden im Folgenden die Eigenschaften von in BPEL4WS abbildbaren Korrelationsprädikaten spezifiziert, sowie für die, die Eigenschaften erfüllenden, Korrelationsprädikate die Abbildung in BPEL4WS charakterisiert.

Anhand der formalen Spezifikation (Abschnitt 5.2.4) von Korrelationsmengen lassen sich die in BPEL4WS abbildbaren Korrelationsprädikate mittels der nachfolgenden Eigenschaften wie folgt charakterisieren:

1. Alle vom Korrelationsprädikat referenzierten Variablen var_i werden im Prozess anhand einer empfangenen Nachricht initialisiert.
2. Alle vom Korrelationsprädikat referenzierten Variablen var_i werden einmal mit Werten belegt.
3. Variablen werden mit dem sie vergleichenden Nachrichtenanteil initialisiert.
4. Für alle von Korrelationsprädikaten referenzierenden Variablen $kpV_i \in Var$ und $kpV_j \in Var$ gilt: $kpV_i \neq kpV_j$ für $i \neq j$ und $kpV_i = kpV_j$ für $i = j$.
5. Es werden ausschließlich Variablen var_i und Nachrichtenteile m_j für die Spezifikation von Vergleichstermen verwendet.
6. Die Menge Rel möglicher Vergleichsoperationen zwischen Variablen var_i und Nachrichtenteilen m_j in Vergleichstermen beschränkt sich auf $Rel = \{=\}$.
7. Die Menge $Bool$ möglicher boolscher Verknüpfungen von Vergleichstermen beschränkt sich auf $Bool = \{\wedge\}$.
8. Für alle von Korrelationsprädikaten referenzierenden Nachrichtenanteile $kpM_i \in MP_M$ und $kpM_j \in MP_M$ gilt: $kpM_i \neq kpM_j$ für $i \neq j$ und $kpM_i = kpM_j$ für $i = j$.

Sind diese Eigenschaften von Korrelationsprädikaten erfüllt, können diese wie folgt als Korrelationsmengen in BPEL4WS abgebildet werden: Für jedes von Korrelationsprädikaten referenzierten Nachrichtenteil wird eine BPEL4WS `property` spezifiziert, deren Name dem Namen des Nachrichtenteil mit dem zusätzlichen Suffix Property entspricht, die anhand von `propertyAlias` dem Nachrichtenteil korrespondierenden Teil, der die Nachricht repräsentierenden WSDL Message zugeordnet wird. Das Korrelationsprädikat wird als Korrelationsmenge (`correlationSet`) abgebildet, die auf die zuvor spezifizierten Propertys referenziert. Die Initialisierung der dem Korrelationsprädikat korrespondierenden Korrelationsmenge im BPEL4WS Prozess erfolgt durch die `receive`-Aktivität, die die korrespondierende, die Variablen initialisierende Nachricht empfängt. Desweiteren wird bei jeder die Nachricht empfangenden ReceiveMessageTask korrespondierenden `receive`-Aktivität die initialisierte Korrelationsmenge zur Korrelierung von Prozessinstanz und Nachricht verwendet.

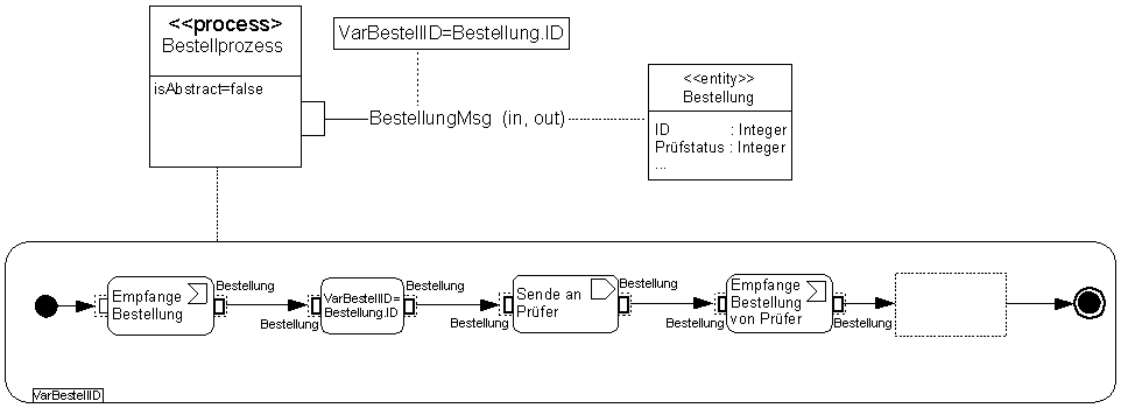

Abbildung 6.10: *Ausschnitt eines Bestell-Prozesse mit einem definierten Korrelationsprädikat.*

Anhand des in Abbildung 6.10 dargestellten Ausschnitts eines *Bestellprozesses* soll die gerade beschriebene Abbildung von Korrelationsprädikaten in BPEL4WS gezeigt werden. Der *Bestellprozess* empfängt zu Beginn eine *Bestellung*, für die ein Korrelationsprädikat spezifiziert ist, das die Variable *VarBestellID* mit der *ID* der *Bestellung* vergleicht. Die Variable *VarBestellID* wird anschließend mit der *ID* der *Bestellung* initialisiert, die daraufhin an einen *Bestellung-Prüfer* versendet wird. Dieser sendet nach dem Prüfen der *Bestellung* diese an den Prozess zurück, wodurch aufgrund der Evaluierung des Korrelationsprädikats diese an den Prozess, der die ursprüngliche *Bestellung* empfangen hat, weitergeleitet wird. Da das Korrelationsprädikat über alle, im vorherigen Absatz spezifizierten Eigenschaften, verfügt, kann es als Korrelationsmenge in BPEL4WS repräsentiert werden. Das nachfolgende Listing zeigt eine schematische Darstellung des BPDM Bestell-Prozesses in BPEL4WS:

```
<message name="BestellungMsg">
    part name="ID"          type="xsd:int"/>
    part name="Pruefstatus" type="xsd:int"/>
    ...
</message>

<bpws:property name="BestellungIDProperty" type="xsd:int"/>
<bpws:propertyAlias propertyName="BestellungIDProperty"
                    messageType="BestellungMsg"
                    part="ID"/>

<process name="Bestellprozess" ... >
    <variables>
        <variable name="VarBestellung" message="BestellungMsg"/>
        <variable name="VarBestellID"  type="VarBestellIDMsg"/>
    </variables>

    <correlationSets>
        <correlationSet name="BestellungIDCorSet" property="BestellungIDProperty"/>
    </correlationSets>

    <sequence>
```

```
        <receive operation="BestellungMsg" variable="VarBestellung" .../>
            <correlation set="BestellungIDCorSet" initate="yes"/>
        </receive>

        <invoke inputVariable="VarBestellung" .../>

        <receive operation="BestellungMsg" variable="VarBestellung" ... >
            <correlation set="BestellungICorSet" initate="no"/>
        </receive>
        ...
    </sequence>
</process>
```

Die *Bestellung* Item Definition wird als `BestellungMsg` WSDL Message abgebildet. Für die Abbildung des Korrelationsprädikates wird eine BPEL4WS Property `BestellungIDProperty` spezifiziert, die anhand eines Property Alias die ID der `BestellungMsg` charakterisiert. Das eigentliche Korrelationsprädikat wird als `correlationSet` mit dem Name `BestellungIDCorSet` abgebildet, welches auf `BestellungIDProperty` referenziert. Da die `VarBestellID` Variable anhand der empfangen Bestellung initialisiert wird, erfolgt die Initialisierung der Korrelationsmenge durch den Empfang der korrespondierten `BestellungMsg` WSDL Message mittels der einleitenden `receive`-Aktivität. Da der „Empfange Bestellung von Prüfer“ Task wiederum auf dem Empfang einer Bestellung wartet, ist die korrespondierende `receive`-Aktivität mit einem zusätzlich `corelation`-Statement versehen, anhand derer die zu empfangende Nachricht mit der entsprechenden Prozessinstanz korreliert wird.

Kapitel 7

Kontrollfluss

Wie bereits in Abschnitt 5.2.5 dargestellt, besteht aufgrund der gegenüber BPEL4WS weniger restriktiven Struktur des Kontrollflusses in BPDM Prozessen das Problem, dass Kontrollflussstrukturen, wie beispielsweise unstrukturierte Schleifen (Abschnitt 5.2.5) und unstrukturierte Verzweigungen (Abschnitt 5.2.5), in BPEL4WS nicht ohne weiteres abgebildet werden können. Daher wird im Folgenden ein Ansatz vorgestellt, der anhand der Beziehungen zwischen Kontrollflussmodellen und Kontrollflussgraphen viele der im Bereich der Programmanalyse und Compiler-Optimierung entwickelten Techniken beispielsweise Kontrollflussnormalisierungsmethoden ([3, 50, 10]) zur Abbildung des Kontrollfluss von BPDM Prozessen in BPEL4WS verwendet [27].

Hierzu werden basierend auf dem in [23] beschriebenen Ansatz zur Analyse von Programm-Kontrollflussgraphen BPDM Prozesse hierarchisch in Regionen, die untereinander unabhängig und jeweils spezifische Kontrollflussstrukturen wie beispielsweise While-Schleifen identifizieren, strukturiert, die anschließend spezifisch bzgl. der sie repräsentierten Kontrollflussstruktur in BPEL4WS abgebildet werden, wobei Regionen, deren Kontrollfluss nicht BPEL4WS konform sind, beispielsweise Regionen mit unstrukturierten Schleifen, vorher durch spezielle auf sie angepasste Restrukturierungs- bzw. Kontrollflussnormalisierungsmethoden gegenüber BPEL4WS restrukturiert werden.

Die Motivation für die Wahl dieses Ansatzes, der abgegrenzte Kontrollfluss-Regionen gegenüber Ansätzen, wie beispielsweise in [27] diskutiert, die den vollständigen Kontrollfluss betrachten, verwendet, liegt hierbei darin, dass zum einen Regionen, die in BPEL4WS als solches nicht abbildbar sind, identifizierbar sind, für Regionen auf diese angepasste Abbildungen spezifiziert werden können, bei denen beispielsweise manuelle Interaktionen mit einem Prozess-Designer möglich sind, sowie damit verbunden zusätzliche globale Analyse-Methoden in einer Divide-and-Conquer Strategie angewendet werden können [23]. Darüber hinaus besteht die Möglichkeit, dass zur Abbildung von Regionen zwischen verschiedenen Abbildungsvarianten ausgewählt werden kann und bei Veränderung der Mächtigkeit der Zielkonstrukte diese sehr leicht angepasst werden können. Desweiteren ist dieser Ansatz neben BPDM auf andere Sprachen, die den in diesem Kapitel spezifizierten Merkmalen bzgl. des damit realisierbaren Kontrollflusses entsprechen, anwendbar.

7.1 Terminologie

Bevor die einzelnen Schritte dieses Ansatzes dargestellt werden, werden im Folgenden einige Begriffe eingeführt, die im weiteren Verlauf dieses Abschnitts zur genaueren Beschreibung benötigt werden und im Kontext von Kontrollflussgraphen charakterisiert sind. Ein Kontrollflussgraph ist hierbei definiert als $G = (N, E, n_S, n_E)$ mit N, der Menge zusammenhängender Knoten, mit $E \subseteq N \times N$, der Menge

Knoten verbindende Kanten, $n_S \in E$ dem Startknoten und $n_E \in E$ dem Endknoten.

Ein Pfad p vom Knoten $x \in N$ zum Knoten $y \in N$ ($x \rightarrow^p y$) ist eine Folge von Knoten $(n_1, ..., n_i)$, $i \in \mathbb{N}$ mit $n_1 = x$ und $n_i = y$, für die gilt, dass $(n_j, n_{j+1}) \in E$ für $j < i$.

Ein Knoten $x \in N$ dominiert einen Knoten $y \in N$ (abk. $d_N(x, y)$), gdw. jeder Pfad von n_S zu y den Knoten x enthält ($\forall n_S \rightarrow^p y : x \in p$). In Analogie zu Knoten läßt sich die Dominanz-Relation auf Kanten übertragen. Sie wird im Folgenden als d_E bezeichnet.

Ein Knoten $x \in N$ postdominiert einen Knoten $y \in N$ (abk. $pd_N(x, y)$), gdw. jeder Pfad von y nach n_E den Knoten x enthält ($\forall y \rightarrow^p n_E : x \in p$). Wie bereits die Dominanz-Relation läßt sich auch die Post-Dominanz-Relation auf Kanten übertragen. Sie wird im Folgenden als pd_E bezeichnet.

Zwei Kanten $x, y \in E$ sind zyklisch äquivalent (abk. $c(x, y)$), gdw. wenn jede Schleife die x enthält auch y enthält und umgekehrt.

7.2 Regionen

Regionen charakterisieren, bezogen auf Kontrollflussgraphen, zusammenhängende Subgraphen $G' = (N', E')$, die aus einer Teilmenge $N' \subseteq N$ von Knoten und zugehörigen Kanten $E' = N' \times N' \cap E$ bestehen, die im Kontext vom BPDM durch Steps und Kontrollflussknoten sowie Daten- und Kontrollflusskanten bestimmt sind.

Da zur Strukturierung von Prozessen im Kontext dieses Ansatzes nicht beliebige Arten von Regionen einsetzbar sind, werden Regionen verwendet, die voneinander unabhängige analysierbare Blöcke charakterisieren, in die Prozesse zerlegbar sind. Hierfür werden die im Kontext der Programmanalyse als Two-Terminal Regionen (TT-Regionen) [27] bezeichneten Regionen genutzt, die über einen Eintrittsknoten $a \in N'$ und einen Austrittsknoten $b \in N'$, die die folgenden Eigenschaften besitzen:

- $d_N(a, k)$ mit $k \in N'$
- $pd_N(b, k)$ mit $k \in N'$

charakterisiert sind. Hierbei gewährleistet die erste Bedingung, dass alle Pfade in die Region über Knoten a führen und die zweite Bedingung, dass alle Pfade aus der Region über Knoten b laufen. In der nachfolgenden Abbildung ist hierzu die Struktur von TT-Regionen schematisch dargestellt.

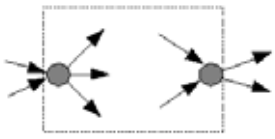

Abbildung 7.1: *Schematische Darstellung von TT-Regionen.*

Bezüglich TT-Regionen existiert eine Sonderform, die lediglich über eine Kante verlassen und eine Kante betreten werden und als Single Entry Single Exit Regionen (SESE-Regionen) [23] bezeichnet werden, die gegenüber TT-Regionen anhand eines Kantenpaares (a, b), $a \neq b$ mit $a \in E$ als Ein- und $b \in E$ als Austrittskante durch folgenden Eigenschaften charakterisierbar sind [23]:

- $d_E(a, b)$

- $pd_E(b, a)$
- $c(a, b)$

wobei die ersten beiden Eigenschaften analog zur Spezifikation von TT-Regionen sind. Zusätzlich gewährleistet die Letztere, dass keine Schleifenkanten in Regionen münden bzw. Regionen verlassen, was bei TT-Regionen bereits durch die Dominanz- und Postdominanz-Eigenschaft der Ein- und Austrittsknoten gewährleistet wird.

Abbildung 7.2: *Äquivalente Transformation von TT- in SESE-Regionen.*

Prinzipiell lassen sich TT-Regionen in SESE-Regionen anhand der in Abbildung 7.2 dargestellte Transformation, bei der für Ein- (N) als auch Austrittsknoten (K) zusätzliche Dummy Knoten (N' bzw. K') erzeugt werden die jeweils alle eingehenden bzw. alle ausgehenden Kanten subsummieren, äquivalent ineinander überführen, wodurch anhand des in [23] beschriebenen Algorithmus zur Ermittlung von SESE-Regionen in $O(|E|)$ im folgenden Ansatz Prozesse anhand von SESE- anstatt TT-Regionen strukturiert werden.

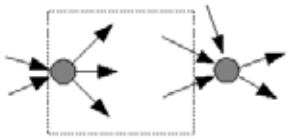

Abbildung 7.3: *Schematische Darstellung von Hammock-Regionen.*

An dieser Stelle sei angemerkt, dass neben TT-Regionen ein ähnliches Konzept, die sog. Hammocks [25], existiert, die ebenfalls Regionen identifizieren, als Subgraphen $G_H = (N_H, E_H, u, v)$ mit $u \in N_H$ (Eintrittsknoten) und $v \in N$ (Austrittsknoten) von G definiert sind, für die gilt, dass alle Kanten von (G/G_H) in G_H über u und alle Kanten von G_H nach (G/G_H) über v gehen [50], die aber aufgrund der Tatsache, dass v außerhalb der Hammock-Regionen liegt und damit das Ziel beliebiger Kanten außerhalb der Regionen sein kann, speziell für den hier vorgestellten Ansatz nicht geeignet sind. Abbildung 7.3 zeigt die schematische Darstellung von Hammock-Regionen.

7.3 Vorgehensweise

Der hier vorgestellte Ansatz zur Abbildung des von BPDM Prozessen spezifizierten Kontrollflusses in BPEL4WS besteht aus 4 Schritten, die gegenüber Prozess-Designern transparent durchführbar sind und im Folgenden näher beschrieben werden:

1. Einleitende äquivalente Transformation.
2. Identifikation von SESE-Regionen.
3. Klassifikation von SESE-Regionen.

4. Restrukturierung und Abbildung der Regionen in BPEL4WS.

Da der hier vorgestellte Ansatz grundsätzlich unabhängig von BPDM Prozessen ist und aufgrund der Vielfalt möglicher Kontrollflussstrukturen zahlreiche Typen an Regionen identifizierbar sind, werden im Folgenden zu dessen weiteren Charakterisierung Prozessflussgraphen (PFG) zur allgemeinen Beschreibung von durch Prozesse spezifizierte Kontrollflüsse, die lediglich aus Verzweigungs- und Verschmelzungsknoten, normalen Knoten, die Arbeitsschritte charakterisieren, Knoten verbindenden Kanten, einem Start- und einem Endknoten bestehen und im wesentlichen Kontrollflussgraphen entsprechen, verwendet. Verschmelzungs- und Verzweigungsknoten werden dabei als Diamant-Symbole dargestellt, normale Knoten als Rechtecke und Start- und Endknoten als ausgefüllte Kreise bzw. ausgefüllte Kreise mit einer zusätzlichen Umrandung. Im Kontext vom BPDM entsprechen hierbei Steps mit mehreren separaten gruppierten Output-Pins Verzweigungsknoten mit einem über eine Kante verbundenen dem Verzweigungsknoten vorangestellten normalen Knoten, der den durch den Step spezifizierten Arbeitsschritt charakterisiert, Steps mit mehreren separat gruppierten Input-Pins Verschmelzungsknoten mit einem über eine Kante verbundenen dem Verschmelzungsknoten nachfolgenden normalen Knoten, Steps normalen Knoten und Start- und FlowFinal-Knoten jeweils Start- und Endknoten.

7.4 Transformation von PFG

Da bei diesem Ansatz die Strukturierung von durch PFGs charakterisierten Prozessen anhand von SESE-Regionen realisiert wird, hierbei aber mögliche Kontrollflussstrukturen gegenüber der Verwendung von TT-Regionen nicht erkannt werden, werden, zur Erkennung weiterer SESE-Regionen, PFGs zu Beginn äquivalent transformiert.

Im ersten Schritt werden hierbei zur Identifikation von Schleifen die Dominanz-Relationen, die ebenfalls in nachfolgenden Schritten benötigt werden, zwischen allen Knoten bestimmt, da gilt, dass eine Kante $x \in E$ mit $x = (x_Q, x_Z)$ eine Schleifenkante ist, gdw. $d_N(x_Q, x_Z)$. Dies kann unter Verwendung des Lengauer-Trajan-Algorithmus in $O(ElogN)$ [28] durchgeführt werden.

Anschließend werden alle kombinierten Verzweigungs-/Verschmelzungsknoten $k \in N$ ermittelt, für deren Menge an Eingangskanten $EK_K = \{x | x \in E \wedge x = (y,k) \wedge x \notin SEK_K\}$, Ausgangskanten $AK = \{x | x \in E \wedge x = (k,y) \wedge x \notin SAK_K\}$, Schleifeneingangskanten $SEK_K = \{x | x \in E \wedge x = (y,k) \wedge d_N(k,y)\}$ und Schleifenausgangskanten $SAK_K = \{x | x \in E \wedge x = (k,y) \wedge d_N(y,k)\}$ gilt: $|EK_K| > 1 \vee |AK_K| > 1 \vee |SEK_K| + |SAK_K| > 1$, und die wie folgt restrukturiert werden:

- Falls $|EK_K| > 1$ wird ein zusätzlicher Verschmelzungsknoten erzeugt, in den alle Eingangskanten EK_K münden, der mit k über eine Kanten verbunden wird. Zusätzlich eingeführte Verschmelzungsknoten können zur Wahrung der Äquivalenz zu PFGs im BPDM in Form von Dummyknoten (empty Steps) realisiert werden, die über eine entsprechende Anzahl an Input/Output-Pins verfügen. Dies kommt aufgrund der spezifizierten Abbildung BPDM nach PFG jedoch an dieser Stelle nicht vor und braucht auch bei der reinen Abbildung von BPDM nach BPEL4WS auf Grund der Transparenz nicht berücksichtigt zu werden.

- Falls $|AK_K| > 1$ wird ein zusätzlicher Verzweigungsknoten erzeugt, aus den alle Ausgangskanten AK_K ausgehen, der mit k über eine in den Verzweigungsknoten eingehenden Kante verbunden wird. In beiden Fällen werden die jeweiligen Kanten von k gelöscht.

- Falls $|SEK_K| + |SAK_K| = 0$, $|EK_K| > 1$ und $|AK_K| > 1$, wird der Knoten k gelöscht.

- Für den Fall, dass $|SEK_K| + |SAK_K| > 1$ und $|SEK_K| >= 1$ und $|SAK_K| >= 1$ wird für alle Schleifeneingangskanten SEK_K ein zusätzlicher Schleifen-Verschmelzungsknoten in Analogie

zur Behandlung von $|EK_K| > 1$, der k vorangestellt wird, erzeugt und die entsprechenden Kanten bzgl. k gelöscht. Zur Gewährleistung der Äquivalenz zu BPDM muss ein Dummyknoten eingeführt werden.

Abbildung 7.4 zeigt hierzu das allgemeine Schema der Restrukturierung von kombinierten Verschmelzungs- / Verzweigungsknoten.

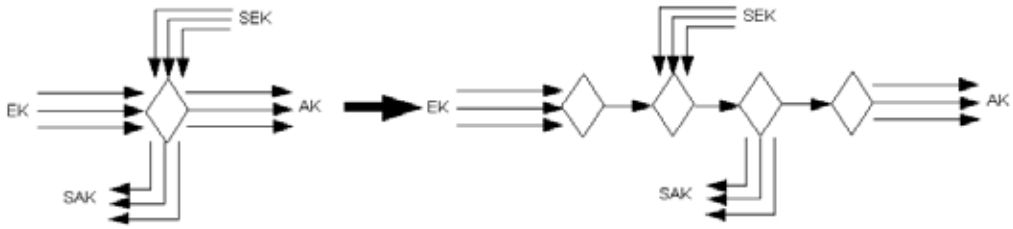

Abbildung 7.4: *Schema der Restrukturierung kombinierter Verschmelzungs- und Verzweigungsknoten.*

Zusätzlich zur Restrukturierung von kombinierten Verschmelzungs-/Verzweigungsknoten können die daraus resultierenden Schleifen-Verschmelzungs- bzw. -Verzweigungsknoten weiter restrukturiert werden falls $|SEK_K| > 1$ oder $|SAK_K| > 1$, um zusätzliche Schleifen als SESE-Regionen zu erkennen. Hierbei wird speziell die Tatsache ausgenutzt, dass Schleifen, die durch Schleifenkanten $x = (x_Q, x_Z) \in E$, deren Schleifenquellknoten x_Q nicht in unterschiedlichen Verzweigungspfaden liegen, ebenfalls TT-Regionen erzeugen, die anderen jedoch nicht. Abbildung 7.5 zeigt das allgemeine Schema der Restrukturierung anhand eines Schleifen-Verschmelzungsknoten, der aus der Anwendung der vorherigen Transformation entstanden ist.

Im Folgenden werden die einzelnen Schritte der Transformation anhand von Schleifen-Verschmelzungsknoten charakterisiert, die in analoger Weise für Schleifen-Verzweigungsknoten durch Vertauschen von Schleifen-Quell- und -Ziel-Knoten und der Erzeugung von Verzweigungs- anstatt Verschmelzungsknoten realisiert werden kann:

- Bestimme für alle Schleifen, die in den Schleifen-Verschmelzungsknoten münden, die Menge SQK der jeweiligen Schleifenquellknoten SQK_i
- Traversiere alle Pfade p_i bzgl. Schleifenquellknoten vom Schleifenverschmelzungsknoten, bis entweder alle SQK_i erreicht wurden oder alle Pfade ab einer bestimmten Stelle übereinstimmen

  Daraus ergibt sich die folgende Struktur:
  $$\begin{array}{ll} p_1: & SQK_i, \ldots, SQK_j \\ p_2: & SQK_k, \ldots, SQK_m \\ \vdots & \vdots \\ p_n: & SQK_n, \ldots, SQK_o \end{array}$$
- Bestimme die in allen Pfaden vorkommenden Schleifenquellknoten und markiere diese.
- Erzeuge eine sortierte Liste der markierten Schleifenquellknoten anhand der bereits bestimmten Dominanz-Relationen.
- Sortiere die in den einzelnen Pfaden zwischen den markierten Schleifenquellknoten enthaltenen nicht markierten Schleifenquellknoten, ohne Duplikate, zwischen den entsprechenden markierten Schleifenquellknoten in die Liste ein.

- Traversiere die sortierte Liste von hinten nach vorne und erzeuge für jeden auftretenden markierten Schleifenquellknoten einen neuen Schleifenverschmelzungsknoten, für alle zwischen markierten Schleifenquellknoten nicht markierten Schleifenquellknoten einen Verschmelzungsknoten, der das Ziel dieser Schleifen wird.

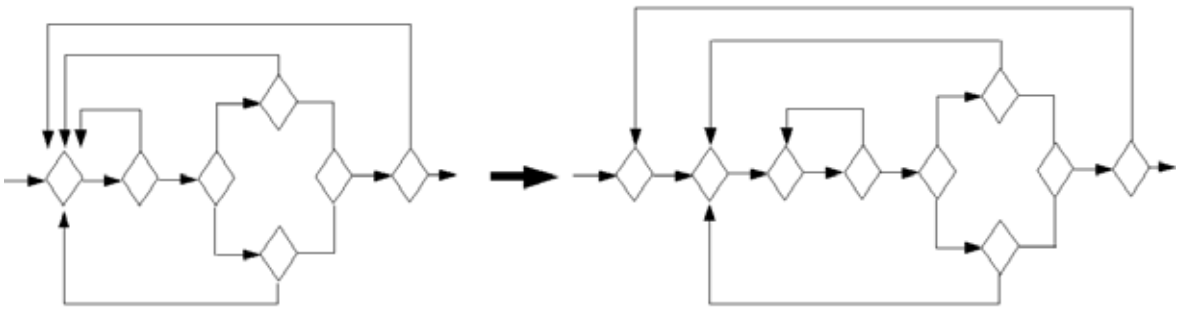

Abbildung 7.5: *Schema der Restrukturierung von Schleifen-Verschmelzungs- und -Verzweigungsknoten.*

Zusätzlich erfolgt die Transformation von Schleifen, die die in Abbildung 7.6 dargestellte Gestalt besitzen, in klassische Until-Schleifen. Dies kann beispielsweise parallel zum Prozess der Restrukturierung kombinierter Verschmelzungs-/Verzweigungsknoten realisiert werden. Im Kontext der Abbildung von BPDM Prozessen ist diese Transformation jedoch überflüssig, da aufgrund der spezifizierten Abbildung zwischen dem Schleifenquell- und dem Verzweigungsknoten mindestens ein zusätzlicher normaler Knoten existiert.

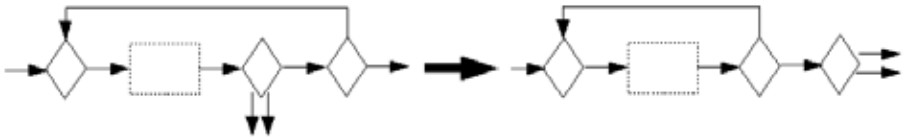

Abbildung 7.6: *Restrukturierung von Until-artigen Schleifen in Until Schleifen.*

7.5 Identifikation von SESE-Regionen

Im zweiten Schritt werden die Prozesse in SESE-Regionen zerlegt, wobei hierbei nicht alle SESE-Regionen ermittelt werden, sondern jeweils die minimalen (sog. kanonischen) SESE-Regionen [23], um eine eindeutig disjunkte hierachische Zerlegung von Prozessen zu ermöglichen, die bei Ermittlung aller SESE-Regionen, wie Abbildung 7.7 schematisch anhand einer einfachen Sequenz von Knoten zeigt, nicht realisierbar wäre. Diese hierarchische Zerlegung kann anhand einer Baumstruktur, die in [23] als Program Structure Tree (PST) bezeichnet wird, deren Wurzel den Prozess, deren Blattknoten dessen Modellelemente und deren inneren Knoten jeweils Regionen darstellen, repräsentiert werden, die im Kontext der algorithmischen Umsetzung als Datenstruktur zur Verwaltung und Traversierung der in Regionen zerlegten Prozesse verwendet werden kann.

Prinzipiell lassen sich SESE-Regionen anhand eines naiven (brute force) Algorithmus, der für jede

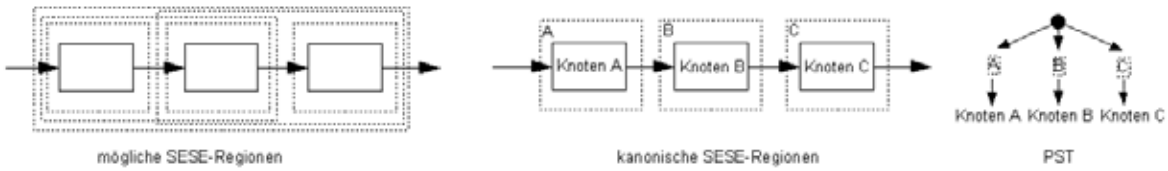

Abbildung 7.7: *Mögliche und kanonische SESE-Regionen sowie deren PST Darstellung.*

Kombination von Kanten die dafür notwendigen Eigenschaften überprüft und damit eine Komplexität von $O(|E|^2)$ besitzt, ermitteln, wodurch ebenfalls TT-Regionen zur Strukturierung verwendet werden können. In [23] findet sich jedoch ein Algorithmus, mit dem aufgrund der Eigenschaft, dass Kanten a, b eine SESE-Region einschliessen, gdw. $c(a, b)$ in $G' = (N, E', n_S, n_E)$ mit $E' = E \cup (n_E, n_S)$, SESE-Regionen und der zugehörige PST in $O(|E|)$ anhand einer Depth-First Traversierung von G ermittelt werden können, der damit bezüglich einer Implementierung zu bevorzugen ist. Für eine genaue Charakterisierung dieses Algorithmus sei an dieser Stelle auf [23] verwiesen.

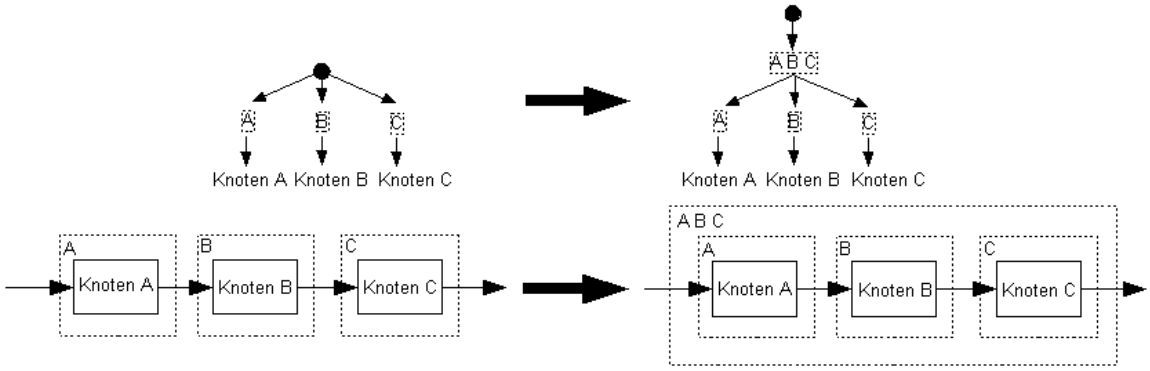

Abbildung 7.8: *Merging von Regionen im PST sowie im BPDM Prozess.*

Da aufgrund der Ermittlung kanonischer SESE-Regionen Sequenzen von Regionen nicht in einer Region subsummiert werden, dies aber für die Klassifikation bzgl. der Charakterisierung von Sequenzen durchaus sinnvoll erscheint, wird der PST bzw. damit verbunden die Prozessstrukturierung anhand eines Mergings von Regionen, das in Abbildung 7.8 dargestellt ist, zusätzlich angepasst. Hierbei werden untereinander benachbarte Regionen, d.h. Regionen deren Eingangskante die Ausgangskante einer anderen Region ist, im PST in einer sie umfassenden maximalen Region subsummiert, indem für diese Regionen im PST eine neue Region erzeugt wird, die die subsummierten Regionen als Nachfolger besitzt. Dieser Vorgang kann mit linearer Komplexität bezüglich der Anzahl der Elemente im PST durchgeführt werden.

7.6 Klassifikation von SESE-Regionen

Im darauffolgenden Schritt werden die identifizierten SESE-Regionen anhand der sie repräsentierenden Kontrollflussstruktur typisiert, mittels dessen im nachfolgenden Schritt die einzelnen Regionen in BPEL4WS geeignet abgebildet werden. Hierfür wird im ersten Teil dieses Abschnitts das zugrundeliegende Klassifikationsschema sowie im darauffolgenden Abschnitt ein Algorithmus zur Bestimmung des Typs (Klasse) von Regionen spezifiziert.

Klassifikationsschema

Grundsätzlich sind Klassifikationen von Regionen von der Vielfalt an Elementen zur Beschreibung des Kontrollflusses der jeweiligen Sprache abhängig, weshalb zu Beginn die Menge möglicher Prozesse auf die beschränkt wurde, die mit Prozessflussgraphen bzgl. ihres spezifizierten Kontrollflusses äquivalent sind, um den Rahmen dieser Arbeit nicht zu sprengen. Trotz dieser Einschränkung sind zahlreiche Klassen an Regionen identifizierbar, für deren Klassifikation aufgrund der Äquivalenz zu Kontrollflussgraphen teilweise Eigenschaften, die speziell im Kontext der Compiler-Theorie verwendet werden, genutzt werden. Im Folgenden werden die einzelnen Klassen des in Abbildung 7.9 dargestellten Klassifikationsschemas näher charakterisiert.

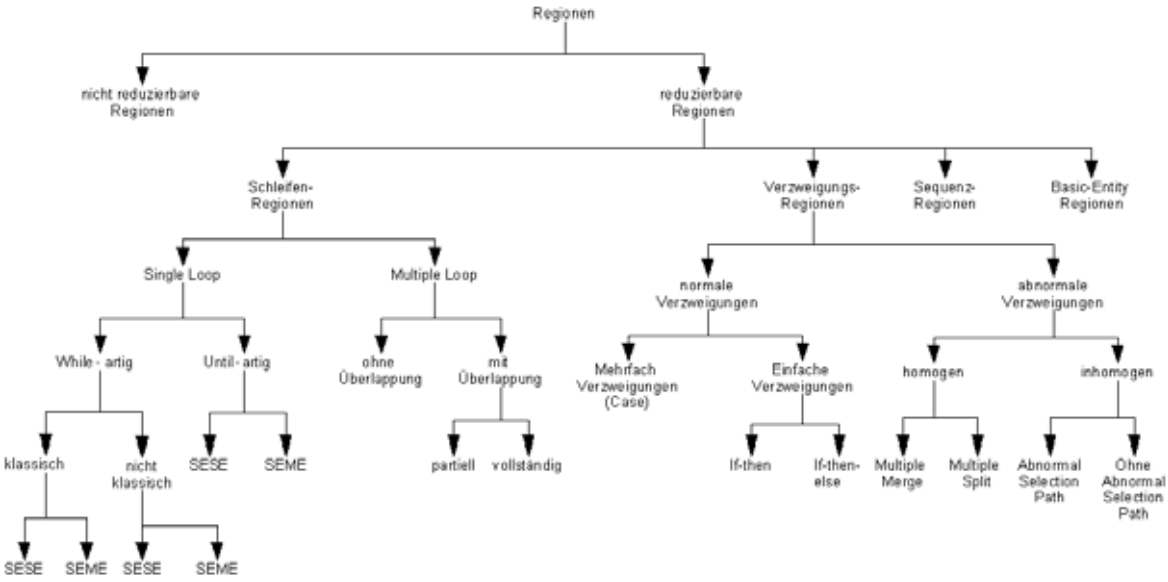

Abbildung 7.9: *Klassifkation von SESE-Regionen.*

Regionen lassen sich aufgrund der Äquivalenz zu Kontrollflussgraphen anhand der Reduzierbarkeit-Eigenschaft, die in der Compiler-Theorie eine sehr wichtige Rolle spielt [1], in zwei große Klassen, zum einen reduzierbare und nicht-reduzierbare Regionen, unterteilen. Hierbei charakterisieren reduzierbare Regionen Regionen, deren Kontrollfluss wohl-strukturiert im Sinne der Partitionierung in zwei disjunkte Mengen, der Menge der Schleifenkanten (Back-Edge), deren Ziel- den Quell-Knoten dominiert und der Menge der Forward-Kanten, die aus dem Rest der vorhanden Kanten bestehen und keine Back-Edge Kanten darstellen [22], zerlegbar sind und Kontrollflussstrukturen charakterisieren, die entweder azyklisch [22] oder zyklisch im Sinne, dass Schleifen durch Back-Edge Kanten realisiert werden, die lediglich multiple Eintrittspfade durch andere in den Schleifenkörper einmündenden Schleifenkanten besitzen können, sind. Demgegenüber sind nicht-reduzierbare Regionen Regionen, die zyklische Kontrollflussstrukturen enthalten, deren Schleifen nicht durch Schleifenkanten realisiert werden. Abbildung 7.10 zeigt schematisch zum Vergleich eine nicht-reduzierbare und reduzierbare Region.

Reduzierbaren Regionen werden weiter in Schleifen-, Verzweigungs-, Sequenz- und Basic-Entity--Regionen unterteilt. Hierbei charakterisieren Basic-Entity-Regionen Regionen, die lediglich einen normalen Knoten im Kontext von BPDM einen Step enthalten, Sequenz-Regionen Regionen, die mehrere Regionen sequentiell abarbeiten, Verzweigungs-Regionen Regionen, die lediglich Verschmelzungs-, Verzweigungsknoten und weitere Regionen beinhalten und Schleifen-Regionen, die Schleifen, die aus-

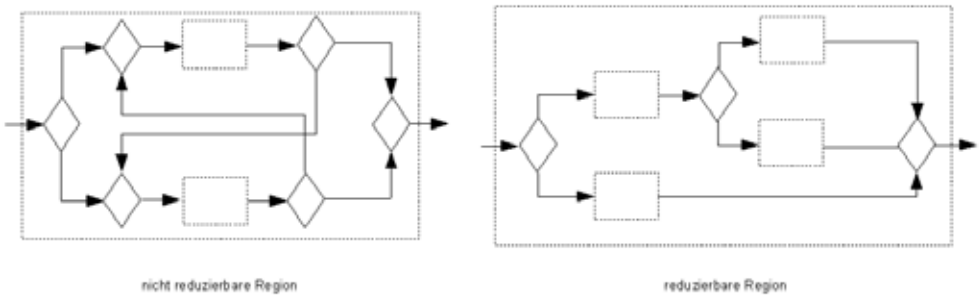

Abbildung 7.10: *Schematische Darstellung einer nicht reduzierbaren und reduzierbaren Region.*

schließlich durch Schleifenkanten, deren Ziel- den Quell-Knoten dominiert, charakterisiert sind. In Abbildung 7.8 sind Basic-Entity-Regionen (A,B,C) sowie eine Sequenz-Region (ABC) erkannbar.

Verzweigungs-Regionen werden in normale und abnormale unterteilt. Hierbei charakterisieren normale Verzweigungs-Regionen Regionen, die lediglich aus einem Verzweigungs- und Verschmelzungsknoten und zusätzlichen Regionen bestehen und damit einen strukturierten Charakter besitzen, wohingegen abnormale Verzweigungs-Regionen mehrere Verzweigungsknoten als auch Verschmelzungsknoten beinhalten und gegenüber normalen Verzweigungs-Regionen als unstrukturiert angesehen werden können.

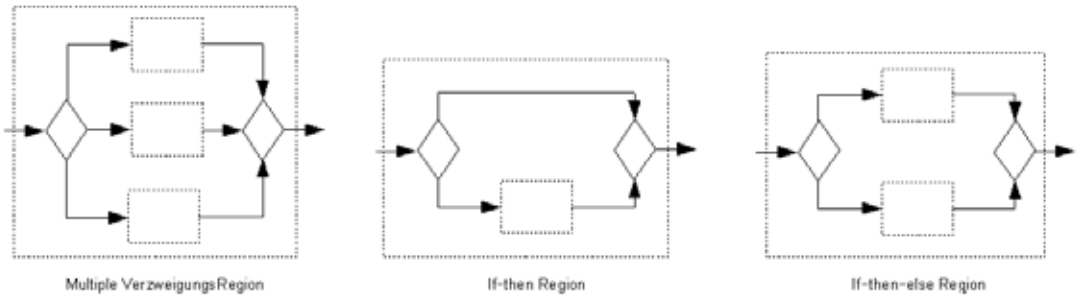

Abbildung 7.11: *Schematische Darstellung einer Mehrfach-Verzweigungs-, If-then- und If-then-else Region.*

Normale Verzweigungs-Regionen werden weiter in Mehrfach-Verzweigungs-Regionen, bei denen der Verzweigungsknoten mehr als zwei ausgehenden Pfade besitzt, was einem CASE-Statement als Kontrollflussstruktur entspricht, und Einfach-Verzweigungs-Regionen, deren Verzweigungsknoten lediglich zwei ausgehende Pfade besitzt, unterteilt, wobei bei Einfach-Verzweigungs-Regionen zusätzlich zwischen If-then Regionen und If-then-else Regionen unterschieden wird. Abbildung 7.11 zeigt schematisch die Struktur der einzelnen Klassen normaler Verzweigungs-Regionen.

Demgegenüber werden abnormale Verzweigungs-Regionen in homogene- und inhomogene Regionen unterteilt. Hierbei charakterisieren homogene bzw. inhomogene Regionen Regionen, die abgesehen auf den einleitenden Verzweigungs- bzw. ausführenden Verschmelzungsknoten entweder nur Verzweigungs- oder Verschmelzungsknoten bzw. Verzweigungs- als auch Verschmelzungsknoten und zusätzliche Regionen enthalten.

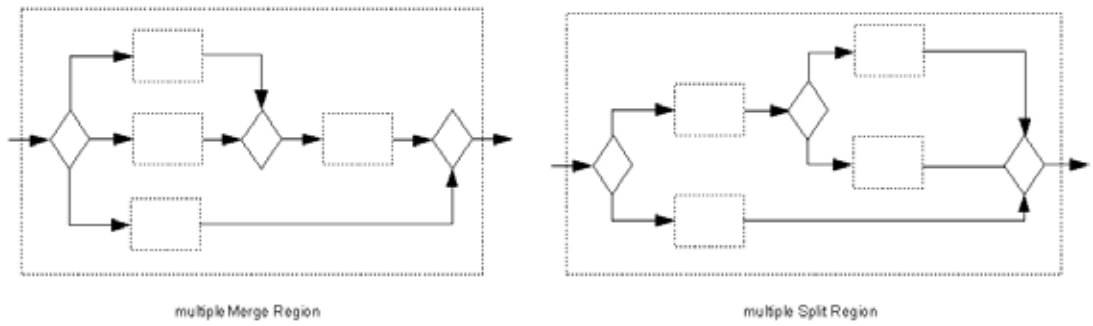

Abbildung 7.12: *Schematische Darstellung einer Multiple Merge- und -Split-Region.*

Homogenen Regionen werden hierbei in multiple-Merge- und multiple-Split-Regionen, die abgesehen auf den einleitenden Verzweigungs- bzw. ausführenden Verschmelzungsknoten entweder nur Verschmelzungsknoten oder Verzweigungsknoten enthalten, unterteilt. Abbildung 7.12 zeigt hierbei schematisch multiple-Merge- und multiple-Split-Regionen.

Demgegenüber werden inhomogene Regionen in Regionen mit und ohne abnormalen Selection Path [46] klassifiziert, wobei Regionen mit abnormalen Selection Path Verschmelzungsknoten enthalten, die gegenüber Regionen ohne abnormalen Selection Path Pfade unterschiedlicher Verzweigungsknoten miteinander zusammenführen. Abbildung 7.15 zeigt hierzu schematisch die Struktur von Regionen mit und ohne abnormalen Selection Path.

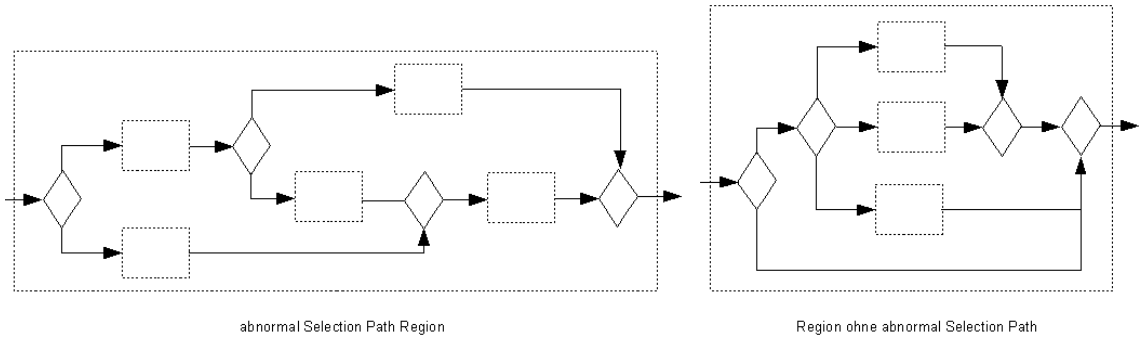

Abbildung 7.13: *Schematische Darstellung einer Region mit abnormalen Selection Path und einer Region ohne abnormal Selection Path.*

Schleifen Regionen werden in Single Loop Regionen, in denen genau eine Schleife vorkommt, und Multi Loop Regionen, die mehrere Schleifen enthalten, klassifiziert. Hierbei werden Single Loop Regionen in While-artige, deren Schleifenquellknoten lediglich eine die Schleifenkante als ausgehende Kante besitzt, und Until-artige, aufgrund der zu Beginn durchgeführten Restrukturierung, deren Schleifenquellknoten neben der Schleifenkante eine weitere ausgehende Kante besitzt, unterteilt. Until-artige Regionen werden hierbei in Schleifenregionen, die lediglich über einen Austrittsspfad (SESE Until Regionen), die den klassischen Until Schleifen entsprechen, und Regionen die über mehrere alternative Austrittspfade (Single Entry Multiple Exit Until Regionen) verlassen werden können, weiter untergliedert. An dieser Stelle sei bermerkt, dass die Klasse der Single Loop Regionen aufgrund der Einordnung in reduzierbare Regionen nur Schleifen mit mehreren Austrittspfaden enthalten kann. Abbildung 7.14

zeigt schematisch SESE- und SEME Until-artige Regionen.

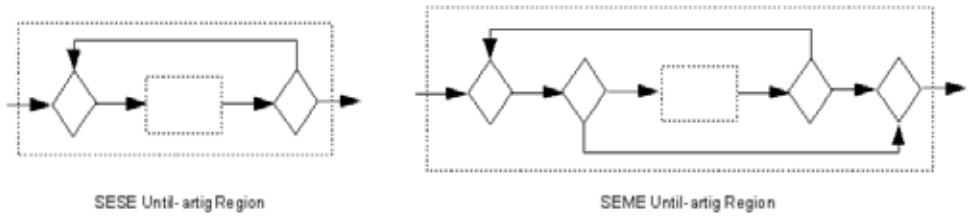

Abbildung 7.14: *Schematische Darstellung einer SESE- und SEME Until-artigen Region.*

Neben Until-artigen Regionen werden While-artige Regionen in klassische, deren Schleifenzielknoten der Verzweigungsknoten, dessen Pfad die Region verlässt, folgt, und nicht-klassische, deren Schleifenzielknoten kein Verzweigungsknoten, dessen Pfad die Region verlässt, folgt, unterteilt. Bei klassischen als auch nicht klassischen Regionen wird wiederum zwischen SESE und SEME Regionen unterschieden. Die nachfolgende Abbildung zeigt schematisch die Struktur der einzelnen Klassen.

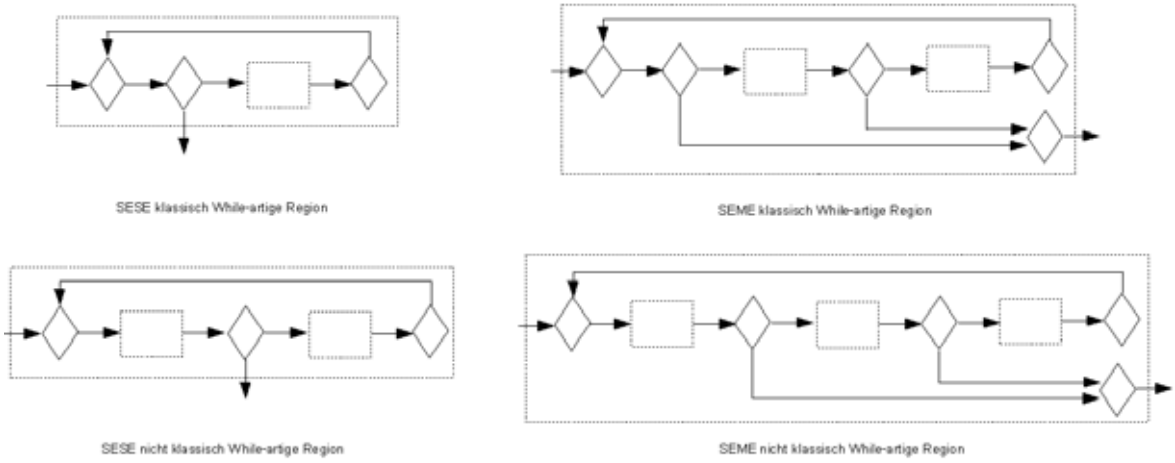

Abbildung 7.15: *Schematische Darstellung einer SESE- und SEME klassisch, nicht klassischen While-artigen Region.*

Gegenüber Single Loop Regionen werden Multi Loop Regionen in Regionen mit überlappenden Schleifen oder vollständig nicht überlappenden Schleifen weiter untergliedert, wobei bei Regionen mit überlappenden Schleifen zwischen Regionen, in denen jede Schleife eine überlappende Schleife darstellt, und Regionen, in denen nur eine teilweise Überlappung vorliegt, unterschieden wird. Da speziell in allen diesen Klassen wiederum zwischen SEME Until-artigen- oder SEME While-artigen Schleifen, die zusätzlich beispielsweise ineinander eingebettet oder parallelisiert [46] sein können, unterschieden werden kann und damit viele weitere Klassen möglich sind, wird aufgrund der damit verbundenen Komplexität ab dieser Stelle die Klassifikation nicht weiter vertieft. Abbildung 7.16 zeigt hierzu schematisch die Strukturen der jeweiligen Regionen.

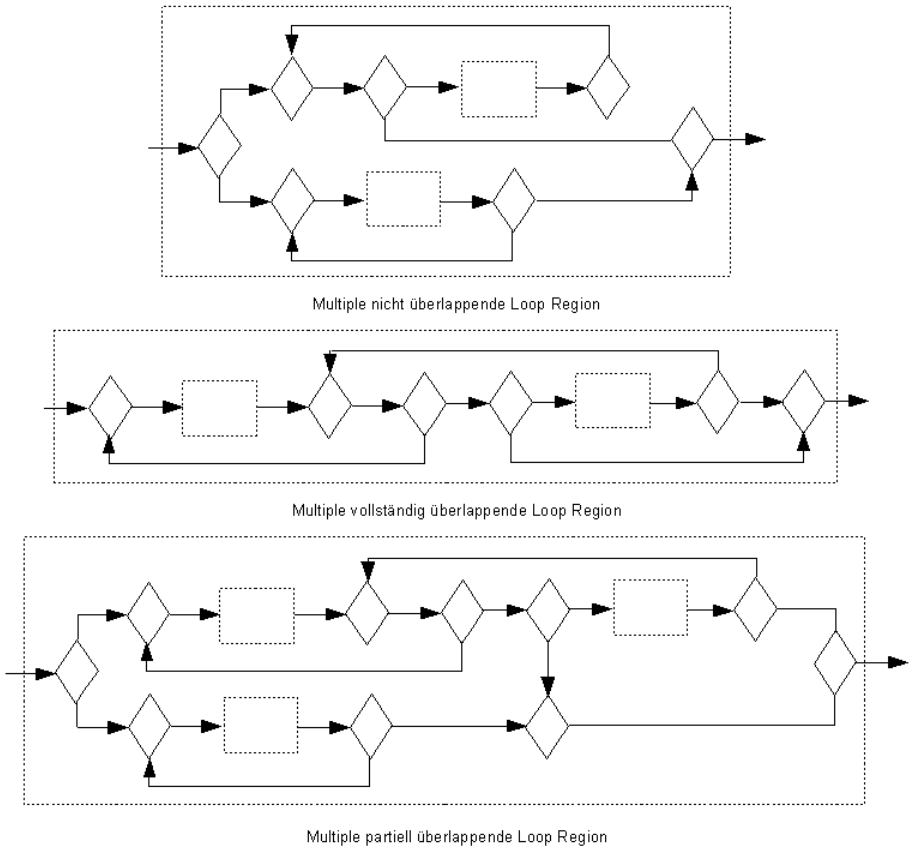

Abbildung 7.16: *Schematische Darstellung nicht-, partiell- und vollständig überlappender Multi Loop Regionen.*

Algorithmus zur Klassifikation von SESE-Regionen

Nachdem das zugrundeliegende Klassifikationsschema charakterisiert wurde, wird im Folgenden ein Algorithmus spezifiziert, der unter Verwendung des PST für jede der identifizierten SESE-Regionen deren Typ unter Durchführung der folgenden Schritte bestimmt:

1. Beginne mit dem der Wurzel nachfolgenden Knoten im PST. Hierbei handelt es sich um eine Region.

2. Bestimme alle Nachfolger-Knoten des aktuellen Knoten im PST.

3. Sind alle Nachfolger-Knoten Regionen, typisiere diese Region als Sequenz-Region und wiederhole Schritt 2 für jede Nachfolger-Region.

4. Besitzt der Knoten jeweils genau einen Nachfolger-Knoten, der keine Region darstellt, dann typisiere die Region als Basic-Entity Region und beende die aktuelle Traversierung.

5. Falls die Menge der Nachfolger-Knoten sowohl Regionen als auch Nicht-Regionen (in diesem Fall Verzweigungs- und Verschmelzungsknoten) enthält, wende die im folgenden beschriebenen Schritte zur weiteren Klassifikation an:

(a) Bestimme die Reduzierbarkeit der Region anhand der in [19] beschriebenen Methode der $T_1 - T_2$ Analyse. Dabei wird der durch die Region charakterisierte Kontrollflussgraph solange durch die Regeln T_1 und T_2 transformiert bis keine weitere Anwendung möglich ist. Ist der daraus resultierende Kontrollflussgraph, der sog. limit flow graph [27], in einem Knoten kollabiert, handelt es sich um einen reduzierbaren Kontrollfuss-Graphen bzw. um eine reduzierbare Region bzw. bei keiner Kollabierung in einem Knoten um eine nicht reduzierbare Region. Die durchzuführenden Transformationen T_1 und T_2 sind hierbei bzgl. des Kontrollflussgraphen $G = (N_R, E_R, s_R, e_R)$ der Region R wie folgt definiert [22]:

- Sei $u \in N_R$. T_1 entfernt die Kante $(u, u) \in E_R$.
- Sei $v \in N_R$ und $v \neq s_R$ und u der alleinige Vorgängerknoten von v. T_2 entfernt den Knoten v und alle Ausgangskanten von v werden Ausgangskanten von u d.h. v wird durch u konsumiert.

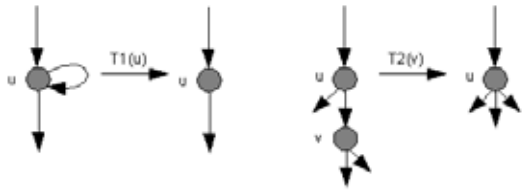

Abbildung 7.17: T_1 *und* T_2 *Transformationen.*

Abbildung 7.17 zeigt schematisch die beiden Transformations-Regeln. Im Falle keiner Kollabierung markiere die Region als nicht reduzierbare Region und führe für alle eingebetteten Regionen Schritt 2 durch.

(b) Bestimme die Anzahl $|T_2|$der Anwendungen von T_2. Gilt dass $|T_2| > 0$, so handelt es sich um eine Schleifen-Region, speziell für $|T_2| = 1$ um eine Single Loop Region bzw. für $|T_2| > 1$ um eine Multi Loop Region, hingegen für $|T_2| = 0$ um eine Verzweigungs-Region. Für Verzweigungs-Regionen fahre mit Schritt 5c fort, für Single Loop Regionen mit Schritt 5f und für Multiple Loop Regionen mit Schritt 5i.

(c) Bestimme die Menge VZK an Verzweigung- und VSK an Verschmelzungsknoten in der Region. Gilt, dass $|VZK| = 1 \wedge |VSK| = 1$, so handelt es sich um eine normale bzw. bei $|VZK| > 1 \vee |VSK| > 1$ um eine abnormale Verzweigungs-Region, speziell für $|VZK| > 1 \wedge |VSK| > 1$ um eine inhomogene bzw. für $(|VZK| > 1 \wedge |VSK| = 1) \vee (|VZK| = 1 \wedge |VSK| > 1)$ um eine homogene multiple Split bzw. multiple Merge Verzweigungs-Region. Im Falle einer homogenen Verzweigungs-Region fahre für alle eingebetteten Regionen mit Schritt 2 fort, für normale Verzweigungs-Regionen mit Schritt 5d und für inhomogene Verzweigungs-Regionen mit Schritt 5e.

(d) Bestimme die Anzahl ak ausgehender Kanten des Verzweigungsknotens. Gilt $ak > 2$, handelt es sich um eine Mehrfach-Verzweigungs-Region, hingegen bei $ak = 2$ um eine einfache Verzweigungs-Region. Speziell für einfache Verzweigungs-Regionen überprüfe, ob eine der Ausgangskanten Eingangskante des Verschmelzungsknotens ist. Ist dem so, handelt es sich um eine If-then Verzweigungs-Region bzw. um eine If-then-else Verzweigungs-Region. Führe für alle eingebetteten Regionen Schritt 2 durch.

(e) Bilde die Menge $\overline{VSK} = VSK/\{k\}$, wobei k den Region-Ausgangs-Verschmelzungsknoten charakterisiert, dessen Ausgangskante die SESE-Regionen mit spezifiziert. Traversiere von jedem Verschmelzungsknoten $vsk_i \in \overline{VSK}$ die Pfade $p_{vsk_i,j}$ zu dem Verzweigungsknoten, in dem alle Pfade zusammenlaufen in umgekehrter Richtung. Bestimme dabei jeweils Menge

$VZK_{p_{vsk_i,j}}$ der traversierten Verzweigungsknoten. Gilt $VZK_{p_{vsk_i,j}} = VZK_{p_{vsk_i,k}}, j \neq k$ für alle $vsk_i \in \overline{VSK}$, so handelt es sich um eine inhomogene Verzweigungs-Region ohne abnormalen Selection Path, ansonsten um eine Region mit abnormalen Selection Path. Führe für alle eingebetteten Regionen Schritt 2 durch.

(f) Bestimme den Schleifenquellknoten sqk der Schleife sowie die Menge AK_{sqk} der ausgehenden Kanten von sqk mit $AK_{sqk} = \{x | x = (sqk, y) \wedge y \in N_R \wedge x \in E_R\}$. Gilt $|AK_{sqk}| = 1$, so handelt es sich um eine While-artige- bzw. bei $|AK_{sqk}| = 2$ um eine Until-artige Schleifen Region. Für While-artige Schleifen Regionen gehe zu Schritt 5h ,für Until-artige zu Schritt 5g.

(g) Ermittle den Region-Ausgangs-Verschmelzungsknoten k und überprüfe ob $k = sqk$ gilt. Wenn ja, handelt es sich um eine SESE- bzw. wenn nein, um eine SEME Until-artige Schleifen-Region. Führe für alle eingebetteten Regionen Schritt 2 durch.

(h) Bestimme den Schleifenkörper $SK_R = (N_{SK}, E_{SK})$ der in der Region enthaltenen Schleife, beispielsweise anhand der Ermittlung aller Pfade $szk \rightarrow^{p_i} sqk$ vom Schleifenzielknoten szk zum Schleifenquellknoten sqk. Bestimme den Nachfolgerknoten $nszk$ von szk. Gilt $AK_{nzsk} \subseteq E_{SK}$, so handelt es sich um eine nicht klassisch bzw. bei $AK_{nzsk} \not\subseteq E_{SK}$ um eine klassisch While-artige Schleifen Region. Für eine klassisch While-artige Schleifen Region bestimme anschließend den Region-Ausgangs-Verschmelzungsknoten k und vergleiche k mit $nszk$. Stimmen beide überein so handelt es sich um eine SESE- bzw. um eine SEME klassisch While-artige Schleifen Region. Für nicht klassisch While-artige Schleifen Regionen bestimme die Menge VKE an Verzweigungsknoten mit $VKE = \{vke | vke \in N_{SK} \wedge \exists x \in N_R : (vke, x) \in E_R\}$. Gilt $|VKE| = 1$ handelt es sich um eine SESE- bzw. für $|VKE| > 1$ um eine SEME nicht klassisch While-artige Schleifen Region. Führe für alle eingebetteten Regionen Schritt 2 durch.

(i) Bestimme die Menge S aller Schleifen innerhalb der Region mit $S = \{(a, b) | d_N(b, a)\}$ unter Verwendung der in Abschnitt 7.4 bestimmten Dominanz-Relationen. Bestimme für alle Schleifen $m = (a_m, b_m)$ in S $ov_m = \{(a, b) | (a \in N_{SK_m} \vee b \in N_{SK_m}) \wedge \neg(a \in N_{SK_m} \wedge b \in N_{SK_m}) \wedge (a, b) \in S\}$ die Menge der mit m überlappenden Schleifen anhand der Traversierung aller Pfade $b_m \rightarrow^{p_i} a_m$, wobei SK_m den Schleifenkörper von m charakterisiert. Gilt für alle $m \in S$, $|ov_m| = 0$, handelt es sich hierbei um eine nicht überlappende Schleifen-Region bzw. falls ein $m \in S$ mit $|ov_m| > 0$ existiert um eine überlappende Schleifen-Region, bei der es sich falls für alle $m \in S$, $|ov_m| > 0$ gilt, um eine vollständig ansonsten um eine partiell überlappende Multiple Schleifen Region handelt.

7.7 BPEL4WS Abbildung

Die Abbildung der in Regionen zerlegten Prozesse in BPEL4WS gestaltet sich, indem alle Regionen, beginnend bei der Wurzel im PST, rekursiv in BPEL4WS abgebildet werden. Hierbei werden in Regionen enthaltene Regionen als atomare Knoten angesehen, die zu Beginn Platzhalter repräsentieren, die im weiteren Prozess durch die Region charakterisierenden äquivalenten BPEL4WS Konstrukte ersetzt werden. Das Ziel der im Folgenden vorgestellten Abbildung ist hierbei, soviel wie möglich BPEL4WS Struktur-Aktivitäten, außer `flow`-Aktivitäten, als Konstrukte zu verwenden (es gibt Gründe, ggfs. einen anderen Ansatz vorzuziehen), um aus Prozessen BPEL4WS Prozesse mit möglichst klar erkennbaren Kontrollflussstrukturen zu erzeugen. Nachdem alle Regionen abgebildet wurden, wird der BPEL4WS Code abschließend weiter optimiert, indem alle überflüssigen `sequence`-Aktivitäten, d.h. zwei direkt ineinander eingebettete `sequence`-Aktivitäten werden in einer subsummiert, entfernt werden.

Aufgrund der Beschränkung von BPEL4WS auf klassische Kontrollflussstrukturen wie Sequenz, Alternative und While-Schleifen lassen sich grundsätzlich nicht alle Typen an Regionen direkt abbilden, wodurch an dieser Stelle zwischen strukturierten und unstrukturierten Regionen unterschieden wird. Strukturierte Regionen charakterisieren hierbei Regionen, die in BPEL4WS direkt, unter Verwendung vorhandener Konstrukte, abbildbar sind, wohingegen unstrukturierte Regionen Regionen spezifizieren, die nicht direkt, sondern indirekt über eine zusätzliche Restrukturierung in BPEL4WS abbildbar sind. Diese Unterscheidung ist prinzipiell für beliebige Zielsprachen möglich, die Zuordnung der einzelnen Typen an Regionen jedoch abhängig von den in der Sprache vorhandenen Modellierungskonstrukten.

Bezogen auf BPEL4WS enthält die Klasse der strukturierten Regionen, auf die im folgenden nicht weiter eingegangen werden soll, Mehrfach-, If-then- und If-then-else Verzweiguns-Regionen, die anhand von `switch`-Aktivitäten realisiert werden, SESE klassisch While-artige Schleifen Regionen, die mittels `while`-Aktivitäten, Sequenz-Regionen, die anhand von `sequence`-Aktivitäten und Basic-Entity-Regionen, die entweder anhand von BPEL4WS Basis-Aktivitäten oder benutzerspezifisch abgebildet werden. Die verbleibenden Klassen an Regionen werden im Kontext von BPEL4WS als unstrukturierte Regionen betrachtet und müssen vor ihrer Abbildung geeignet, funktional äquivalent [27], restrukturiert (gegenüber BPEL4WS normalisiert), im Sinne der Realisierung mit den in BPEL4WS vorhanden klassischen Kontrollflussstrukturen, werden.

Hierbei können aufgrund der Äquivalenz von Kontrollflussmodellen und Kontrollflussgraphen Restrukturierungs-Methoden (Normalisierungsmethoden) aus dem Bereich der Compiler-Optimierung, wie beispielsweise Methoden zur Goto-Elimination von Erosa u. Hendren [15], Armaguellat [3], Zhang u. Hollander [50] basierend auf Hammock-Graphen oder Williams/Ossher zur Strukturierung von Fluss-Diagrammen [46], von denen aufgrund ihres Umfangs und Komplexität in dieser Arbeit nur einige Ausgewählte charakterisiert werden, Anwendung finden, die beispielsweise anhand der Art der Repräsentation des Kontrollflusses in Fluss-Diagramm-, Turing-Maschine-, Programm-, Schema- und Binary-Methoden nach [50] unterschieden werden können. Da der Kontrollfluss von Prozessen anhand von Fluss-Diagrammen spezifiziert wird, werden die in den folgenden Abschnitten beschriebenen Restrukturierungen, vorwiegend als Fluss-Diagramm basierte Methoden, charakterisiert. Jedoch können in Abhängigkeit der bei der Restrukturierung einzuhaltenden Randbedingungen, beispielsweise der Verhinderung von Knoten-Duplizierung oder Generierung möglichst wenig zusätzlicher Variablen, andere Arten von Methoden zur Anwendung kommen, die jedoch eine zusätzliche vorherige Abbildung in die jeweilige Repräsentation vorraussetzen.

Im Folgenden werden für die verschiedenen Typen unstrukturierter Regionen Methoden zur Restrukturierung bzgl. BPEL4WS beschrieben um einen vollständigen Ansatz zur Abbildung des Kontrollflusses von Prozessen in BPEL4WS zu charakterisieren. Hierbei werden Methoden benannt, die grundsätzlich anwendbar sind, jedoch Methoden genauer charakterisiert, die mit Hinblick auf BPEL4WS den Kontrollfluss weitestgehend nachvollziehbar für einen Benutzer restrukturieren, wodurch an diese Stelle Eigenschaften wie die Minimierung von Knoten-Duplikationen oder der Anzahl eingeführter Variablen vernachlässigt werden.

7.7.1 nicht reduzierbare Regionen

Eine Klasse unstrukturierter Regionen sind die nicht reduzierbaren Regionen, die über den Schritt der Transformation in reduzierbare Regionen in BPEL4WS abgebildet werden können. Hierbei können entweder Methoden zur Kontrollfluss-Normalisierung, die nicht reduzierbare Kontrollflussgraphen unterstützen, beispielsweise die in [50, 27, 15, 3, 46] genannten Methoden, wobei [50, 15, 3] Programm- und [27, 46] Fluss-Diagramm basiert sind, oder direkt Methoden zur Transformation nicht reduzierbarer Regionen (Kontrollflussgraphen), in reduzierbare Regionen beispielsweise der in [22] genannte Ansatz des kontrollierten Knoten-Splittens, bei der, nach einer Heuristik, ausgewählte Knoten (Steps)

dupliziert werden, angewendet und die daraus resultierenden Regionen neu strukturiert, klassifiziert und anschließend anhand ihres Typs abgebildet werden. Da speziell der in [22] beschriebene Ansatz zur Transformation nicht reduzierbarer Regionen in reduzierbare die $T_1 - T_2$ Analyse verwendet um die die Irreduzierbarkeit verursachenden Knoten zu identifizieren, kann er speziell den im Schritt der $T_1 - T_2$ Analyse des in Abschnitt 7.6 beschriebenen Algorithmus zur Typisierung von SESE-Regionen bestimmten limit flow graph weiter verwenden ohne zusätzlich eine erneute vollständige $T_1 - T_2$ Analyse durchzuführen.

7.7.2 Multiple Merge- und -Split Regionen

Neben nicht reduzierbaren Regionen gehören ebenfalls Multiple Merge und -Split Regionen in die Klasse der unstrukturierten Regionen. Diese können ebenfalls unter Verwendung der im vorherigen Abschnitt erwähnten Methoden zur Kontrollfluss-Normalisierung restrukturiert werden, die jedoch zu sehr unübersichtlichen BPEL4WS Strukturen, besonders die in [15] beschriebene Methode, führen. Daher wird im Folgenden eine Methode dargestellt, bei der unter gezielter Einführung zusätzlicher Verzweigungs- bzw. Verschmelzungsknoten die zugrundeliegende Struktur weitestgehend erhalten bleibt. An dieser Stelle sei bemerkt, dass Multiple Merge und -Split Regionen in BPEL4WS grundsätzlich mittels `flow`-Aktivitäten ohne vorherige Restrukturierung abgebildet werden können, was aber bei zunehmender Kantenanzahl zu sehr unübersichtlichen Strukturen in XML führt, wodurch dieser Ansatz an dieser Stelle nicht weiter charakterisiert wird.

Diese Methode basiert darauf, dass bei multiplen Merge- bzw. Split Regionen mehrere Verschmelzungs- mit einem Verzweigungsknoten bzw. mehrere Verzweigungs- mit einem Verschmelzungsknoten korrespondieren, so dass zur Restrukturierung lediglich der korrespondierende Verzweigungsknoten in mehrere den Verschmelzungsknoten korrespondierende, äquivalente Verzweigungsknoten bzw. der Verschmelzungsknoten in mehrere den Verzweigungsknoten korrespondierende, äquivalente Verschmelzungs-Knoten zerlegt werden müssen um die Kontrollflussstruktur anhand in BPEL4WS abbildbarer normaler Verzweigungs-Regionen zu realisieren. Abbildung 7.18 zeigt hierzu ein Beispiel der Restrukturierung anhand einer multiplen Merge Region, bei der durch Zerlegung des eingehenden Verzweigungsknoten in zwei Verzweigungsknoten einen für den inneren als auch für den äußeren Verschmelzungsknoten die Region restrukturiert wird.

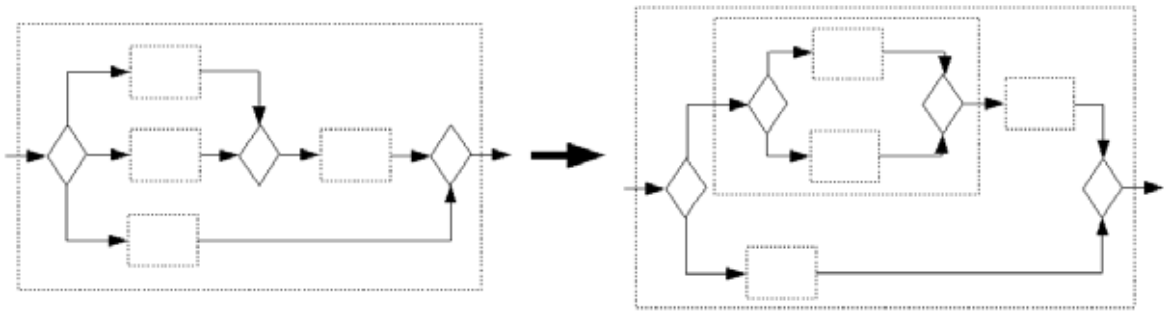

Abbildung 7.18: *Schematische Darstellung der Restrukturierung einer Multiplen Merge Region anhand der Einführung eines zusätzlichen Verzweigungsknotens.*

Der Ablauf der Restrukturierung umfaßt im wesentlichen die folgenden Schritte, charakterisiert anhand der Behandlung von multiple Merge Regionen:

1. Wähle den eingehenden Verzweigungs- VZK und ausgehenden Verschmelzungsknoten VSK der Region aus.

2. Bestimme die Menge EK_{VSK} der eingehenden Kanten des Verschmelzungsknotens VSK und die Menge AK_{VZK} der ausgehenden Kanten des Verzweigungsknotens VZK. Falls $|AK_{VSK}| \neq |EK_{VZK}|$ gehe zu Schritt 3, ansonsten beende die aktuelle Bearbeitung und markiere die Region entweder als If-then-, If-then-else- oder Mehrfach-Verzweigungs-Region anhand der im Klassifikationsalgorithmus spezifizierten Methode.

3. Bestimme zu jeder Eingangskante $e_i \in EK_{VSK}$ die Menge $AK_{VZK,e_i} \subset AK_{VZK}$ der korrespondierenden Ausgangskanten des Verzweigungsknotens, für die gilt: $AK_{VZK} = \bigcup_{j=1}^{n} AK_{VZK,e_j}$ mit $AK_{VZK,e_k} \cap AK_{VZK,e_l} = \emptyset$ für $k \neq l$. Hierbei korrespondiert eine Eingangskante e_i eines Verschmelzungsknotens V mit einer Ausgangskante a_j eines Verzweigungsknotens Z, gdw. ein Pfad von Z nach V existiert, der über die Kanten a_j und e_i verläuft.

4. Erzeuge für jede Eingangskante $e_i \in EK_{VSK}$ mit $|AK_{VZK,e_i}| > 1$ einen neuen Verzweigungsknoten VZK_{e_i}, der als Ausgangskanten AK_{VZK,e_i} sowie eine Eingangskante (VZK, VZK_{e_i}), die als Bedingung $\bigvee_{ak \in AK_{VZK,e_i}} cond(ak)$, wobei *cond* eine Funktion ist, die die jeweilige Bedingung einer, einen Verzweigungsknoten verlassenden, Kante bestimmt, spezifiziert, besitzt. Abschließend werden alle Ausgangskanten AK_{VZK,e_i} von VZK gelöscht. Markiere die Region entweder als If-then, If-then-else oder Mehrfach-Verzweigungs-Region anhand der im Klassifikationsalgorithmus spezifizierten Methode und identifiziere möglicherweise entstandenen Regionen.

5. Führe für die so entstandenen neuen Verzweigungs Regionen Schritt 1 durch.

Für multiple Split Regionen gestaltet sich der Ablauf in ähnlicher Weise.

7.7.3 SESE Until-artige Schleifen Regionen

Da BPEL4WS lediglich ein Konstrukt zur Spezifikation von While- anstatt von Until-Schleifen bietet, fallen ebenfalls SESE Until-artige Schleifen Regionen, die klassische Until-Schleifen charakterisieren, in die Klasse unstrukturierter Regionen.

Aufgrund der Äquivalenz zwischen Until- und While-Schleifen können SESE Until-artige Schleifen Regionen entweder mittels Knoten-Duplikation oder der Einführung einer zusätzlichen „Variable“, die lediglich bezüglich der Abbildung nach BPEL4WS Bedeutung besitzt, in SESE klassisch While-artige Schleifen Regionen restrukturiert und in BPEL4WS abgebildet werden. Abbildung 7.19 zeigt hierzu die beiden Restrukturierungs-Arten.

Bei der Knoten-Duplikation wird die den Schleifenkörper charakterisierende Region kopiert und vor der Until Region eingefügt sowie gleichzeitig ein Verzweigungsknoten, der dem Schleifenzielknoten folgt mit der den Schleifenkörper verlassenden Kante, die entsprechend gelöscht wird, als Ausgangskante erzeugt. Hierfür werden die folgenden Schritte durchgeführt:

1. Bestimme den Schleifenquell- SQK, Schleifenzielknoten SZK, die Bedingung $cond(sk)$ der Schleifenkante $sk = (SQK, SZK)$ sowie die von SQK und SZK umschlossene SESE-Region SK, deren Eintrittskante a die Ausgangskante von SQK und deren Austrittskante b die Eingangskante von SZK ist.

2. Kopiere SK. Ersetze hierbei die kopierte Kante a durch die die SESE-Schleifen Region charakterisierende Eingangskante und setzte das Ziel der Kante b auf SZK.

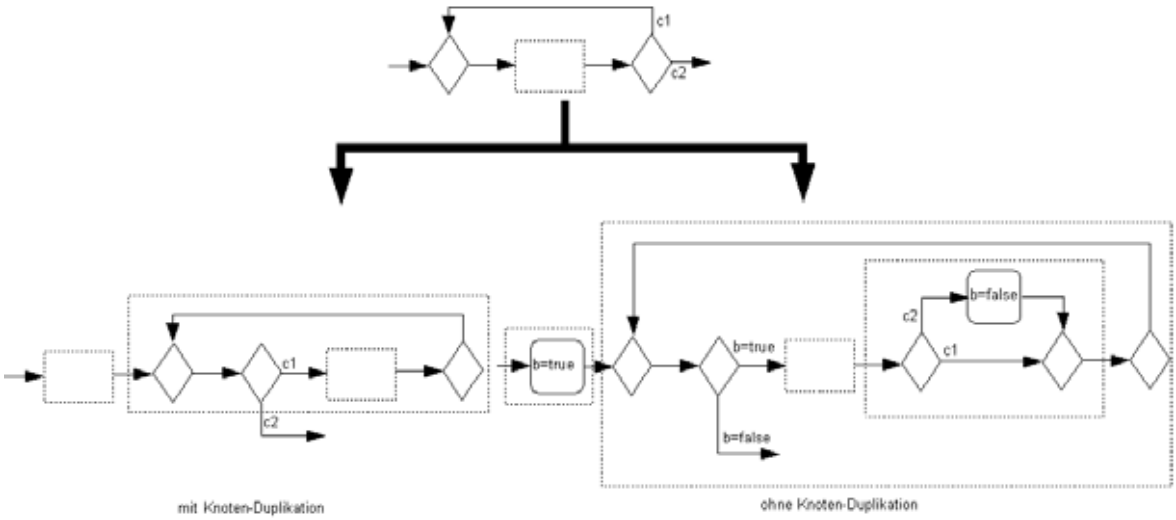

Abbildung 7.19: *Schematische Darstellung der Restrukturierung einer SESE Until-artigen Schleifen Region anhand der Duplikation des Schleifenkörpers oder der Einführung einer zusätzlichen „Variable“.*

3. Füge nach SZK einen Verzweigungsknoten VZK ein, indem der Zielknoten von a VZK wird und eine neue Kante k, mit $k = (VZK, a_Z)$, wobei a_Z den ursprünglichen Zielknoten von a bezeichnet, erzeugt wird, deren Bedingung der Schleifenbedingung $cond(sk)$ entspricht. Zusätzlich wird der Quellknoten der Austrittskante der SESE Schleifen Region auf VZK gesetzt.

4. Erzeuge eine Sequenz-Region, die die ursprüngliche SESE Schleifen Region und SK enthält.

Bei der zweiten Möglichkeit wird, anstatt die den Schleifenlkörper repräsentierte Region zu duplizieren, eine lediglich für die Abbildung nach BPEL4WS (bzw. Zielsprache) relevante „Variable“ die anhand eines dem Schleifenzielknoten vorangehenden Knoten initialisiert wird, eingefügt, die aufgrund des Zustandes der Schleifenaustritts-Bedingung entsprechend mit Werten belegt wird gewährleistet, dass die Schleife mindestens einmal durchlaufen wird. Hierfür werden die folgenden Schritte durchgeführt:

1. Bestimme den Schleifenquell- SQK, Schleifenzielknoten SZK, die Bedingung $cond(sk)$ der Schleifenkante $sk = (SQK, SZK)$ sowie die Bedingung a der die Schleife verlassenden Kante.

2. Füge einen Knoten i ein, der gleichzeitig als Basis-Entity Region typisiert wird, der die neu spezifizierte Variable b mit *wahr* initialisiert. Der Ziel-Knoten der Eingangskante der SESE-Region wird i. Zusätzlich wird eine neue Kante $k = (i, SZK)$ erzeugt.

3. Erzeuge einen Verschmelzungsknoten VSK, einen Verzweigungsknoten VZK sowie einen Knoten j, der b mit falsch belegt. Ersetze das Ziel der Ausgangskante mit der Bedingung a durch j und die Quelle durch VZK. Erzeuge eine Kante von VZK zu VSK mit der Bedingung $cond(sk)$, sowie eine Kante von j zu VSK. Ersetze den Zielknoten der in SQK mündenden Kante durch VZK. Verbinde VSK mit SQK durch eine weitere Kante. Zusätzlich wird die durch VSK und VZK neu entstandene if-then Region markiert.

4. Füge einen neuen Verzweigungsknoten VZK^2 ein. Der Zielknoten der den Schleifenzielknoten verlassenden Kante y wird VZK^2. Gleichzeitig wird eine neue Kante z erzeugt, deren Quellknoten VZK^2 und deren Zielknoten dem von y ursprünglichen Zielknoten entspricht. Die Kanten-Bedingung von z ist $b = true$. Gleichzeitig wird der Quellknoten der die Schleife verlassenden Kante VZK^2 und deren Bedingung durch $b = false$ ersetzt.

7.7.4 SESE nicht klassisch While-artige Schleifen Regionen

Neben SESE Until-artigen Schleifen Regionen gehören auch SESE nicht klassisch While-artige Schleifen Regionen in die Klasse unstrukturierter Regionen, die beispielsweise unter Verwendung der Eingangs erwähnten Kontrollflussnormalisierungs-Methoden in SESE klassisch While-artige Schleifen Regionen restrukturiert werden können. An dieser Stelle soll jedoch eine auf einer bereits beschriebenen Methode basierte Restrukturierungs-Methoden charakterisiert werden.

Hierbei werden SESE nicht klassisch While-artige Schleifen Regionen anhand der Einführung eines zusätzlichen Verschmelzungsknotens und der Verlagerung der die Regionen verlassenden Kante hinsichtlich des Schleifenquellknotens in SESE Until-artige Schleifen Regionen restrukturiert (Abbildung 7.21), die anschließend anhand der im vorherigen Abschnitt charakterisierten Restrukturierung von SESE Until-artigen Schleifen Regionen in SESE klassisch While-artige Schleifen Regionen restrukturiert und in BPEL4WS abgebildet werden.

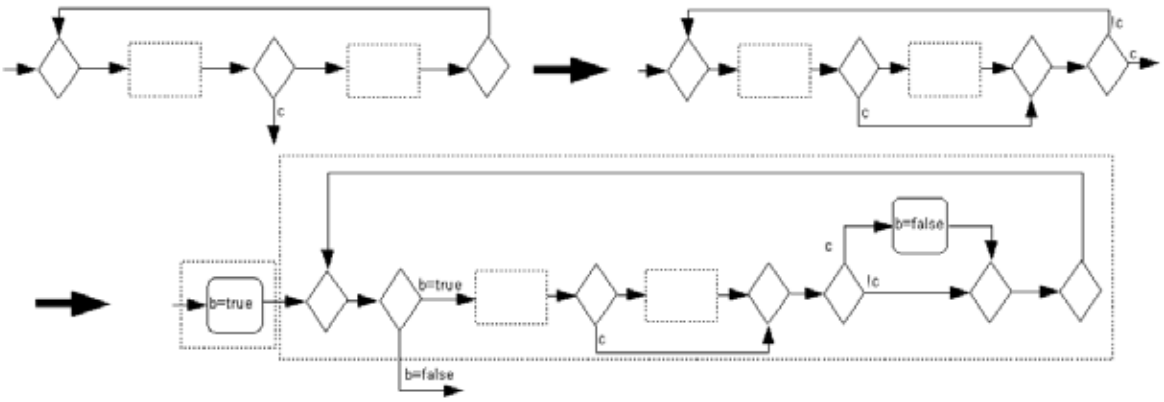

Abbildung 7.20: *Schematische Darstellung der Restrukturierung einer SESE nich klassisch While-artigen Schleifen Region.*

Anhand der nachfolgenden Schritte lässt sich der Ablauf der Restrukturierung wie folgt charakterisieren:

1. Bestimme den Schleifenquellknoten SQK der die Region enthaltenen Schleife, sowie den die Austrittskante a mit der Bedingung $cond(a)$ der SESE-Regionen als Quelle zugeordneten Verzweigungsknoten VZK.
2. Füge einen neuen Verschmelzungsknoten VSK vor SQK ein. Die Eingangskante von SQK wird zur Eingangskante von VSK.
3. Verbinde VSK mit SQK durch eine Kante (VSK, SQK).
4. Ersetze den Zielknoten von a durch VSK.
5. Erzeuge eine Kante (SQK, a_Z) mit a_Z, dem ursprünglichen Zielknoten von a, und der Bedingung $cond(a)$. Gleichzeitig wird die Schleifenkante mit der Bedingung $!cond(a)$ belegt.
6. Identifiziere möglicherweise neu entstandene if-then Region.
7. Wende auf die durch die Restrukturierung entstandene SESE Until-artige Schleifen Region eine der in Abschnitt 7.7.3 genannten Methoden zur Umwandlung der SESE Until- in SESE klassisch While-artige Schleifen Regionen an.

7.7.5 SEME klassisch While-artige Schleifen Regionen

Neben SESE Until-artigen Schleifen- und SESE nicht klassisch While-artige Schleifen Regionen gehören ebenfalls SEME klassisch While-artige Schleifen Regionen in die Klasse der unstrukturierten Regionen, die ebenfalls unter Verwendung der Eingangs erwähnten Ansätze in SESE klassisch While-artige Schleifen Regionen, die in BPEL4WS abbildbar sind, restrukturiert werden können. Aufgrund der zahlreichen Methoden werden im Folgenden zwei, die für die Restrukturierung verwendbar und gegenüber anderen recht kompakt spezifizierbar sind, charakterisiert: die Fluss-Diagramm basierte Methode nach Williams und Osher [46] sowie die Programm-basierte Methode mittels Goto-Elimination nach Erosa und Hendren [15].

Restrukturierung anhand der Methode von Williams und Ossher

Die Methode von Williams und Osher ist eine Fluss-Diagramm basierte Methode zur Restrukturierung von While Schleifen mit mehreren Austrittspfaden, die SEME klassisch While-artigen Schleifen Regionen entsprechen, in While Schleifen mit einem Austrittspfad, äquivalent zu SESE klassisch While-artigen Schleifen Regionen, die damit direkt zur Restrukturierung von SEME klassisch While-artigen Schleifen Regionen verwendbar ist.

Sie restrukturiert SEME klassisch While-artige Schleifen Regionen anhand der Einführung einer zusätzlichen boolschen „Variable“ b, einer ganzzahligen „Variable“ i sowie mehreren Verzweigungs- und Verschmelzunsgknoten ohne zusätzliche Knoten bzw. Region-Duplizierung, indem alle den Schleifenkörper verlassenden Austrittskanten in jeweils einen korrespondierten im Schleifenkörper enthaltenen Veschmelzungsknoten geführt werden, die vor Erreichen des Verschmelzungsknoten jeweils einen Knoten, der i mit dem die Austrittskante charakterisierenden Wert und b mit $false$ belegt, ausführen, so dass beim nächsten Schleifendurchlauf die einzige Austrittskante des Schleifenkörpers, die dem neu eingefügten Verzweigungsknoten mit der Bedingung $b = false$ zugeordnet ist, gewählt wird, deren nachfolgende Verzweigungsknoten mittels i das Verhalten des jeweils gewählten Austrittspfades realisieren. Abbildung 7.21 zeigt schematisch die Restrukturierung.

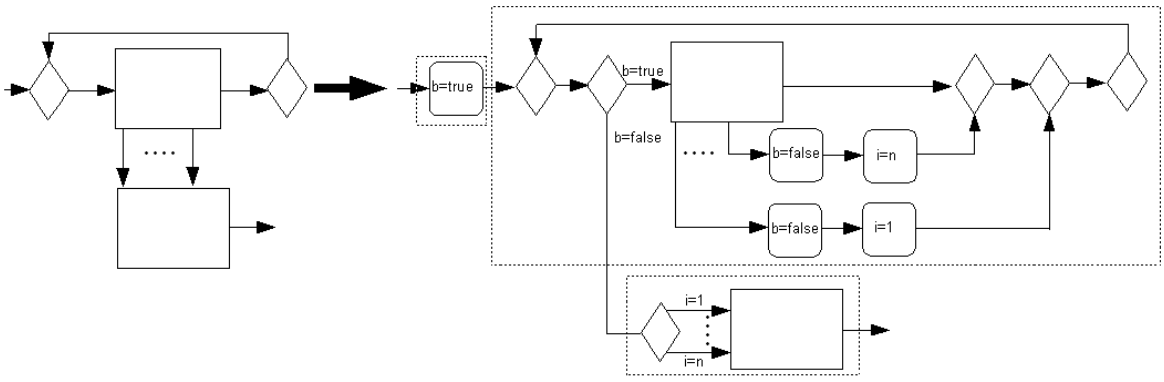

Abbildung 7.21: *Schematische Darstellung der Restrukturierung einer SEME klassisch While-artigen Schleifen Region anhand der Methode von Williams und Ossher.*

Anhand der nachfolgenden Schritte wird der Ablauf der Restrukturierung wie folgt charakterisiert:

1. Bestimme den Schleifenkörper $SK = (N_{SK}, E_{SK})$ der SEME klassisch While-artigen Schleifen Region, wobei N_{SK} die Menge der bei wiederholten Schleifendurchläufen erreichbaren Knoten und E_{SK} Kanten charakterisiert.

2. Bestimme die Menge $AK_{SK} \subset E_R$ den Schleifenkörper verlassenden Austrittskanten mit $AK_{SK} = \{x | x = (x_Q, x_Z) \in E_R \wedge x_Q \in N_{SK} \wedge x_Q \in N_R\}$ und ordne jeder Austrittskante anhand der bijektiven Abbildung $f_{AK} : AK_{SK} \rightarrow N$ eine eindeutige Zahl zu.

3. Füge für jede Austrittskante $a \in AK_{SK}$ eine Veschmelzungsknoten VSK, dem der Schleifenquellknoten folgt, einen Knoten der b mit $false$ und einen Knoten der i mit $f_{AK}(a)$ belegt ein, die sequentiell über Kanten miteinander verbunden sind, wobei der Verschmelzungsknoten der letzte in der Sequenz ist. Der Zielknoten der Austrittskante wird bezüglich des ersten Knotens geändert.

4. Füge dem Schleifenzielknoten nachfolgend einen Verzweigungsknoten (empty Step) ein, der mit diesem über eine Kante verbunden ist und dem zwei Ausgangskanten, die des Schleifenzielknotens, die mit der Bedingung $b = true$ und eine neue k, die mit $b = false$ belegt werden, zugeordnet sind.

5. Füge als Zielknoten von k einen Verzweigungsknoten ein, der $|AK_{SK}|$ Austrittskanten besitzt, die jeweils mit $i = f(a)$ als Bedingung belegt sind und deren Zielknoten dem der ursprünglichen Austrittskante entspricht.

6. Klassifiziere die im inneren der Schleifen Region neu enstandene Verzweigungs-Region.

Restrukturierung anhand der Goto-Elimination nach Erosa und Hendren

Die Methode von Erosa und Hendren ist eine Methode zur Restrukturierung des durch Goto-Anweisungen in C Programmen realisierten Kontrollflusses mittels klassischen C-Kontrollflussstrukturen (`if-then`, `while`, `switch`) durch die äquivalente Substitution von Goto-Anweisungen mit diesen. Da mittels Goto-Anweisungen beliebige Kontrollflussstrukturen, darunter auch die durch SEME klassisch While-artige Schleifen Regionen beschriebene, spezifizierbar sind und klassische Kontrollflussstrukturen äquivalente strukturierte Regionen charakterisieren, kann diese Methode ebenfalls zur Restrukturierung verwendet werden.

Da es sich hierbei jedoch um eine Programm-basierte Methode handelt, muss zur Anwendung der Kontrollfluss der jeweiligen zu restrukturierenden Regionen zuvor in äquivalente C-Notation überführt werden. Hierzu werden Kanten durch Goto-Statements, Knoten bzw. Regionen durch Prozeduraufrufe und While Schleifen durch While-Anweisungen charakterisiert. Die nach durchgeführter Restrukturierung durchzuführende Rücktransformation erfolgt, indem für Kontrollflussstrukturen und Procedureaufrufe seperate äquivalente Regionen erzeugt werden die jeweils mit der ihnen nachfolgenden Region über eine Kante verbunden und in ihrer Hierachisierung bzgl. den Kontrollflussstrukturen äquivalent sind.

Der von M. Erosa beschriebene Ansatz zur Goto-Eliminierung in klassischen C Programmen basiert auf der Anwendung einer Menge von äquivalenten Programm-Transformationen, die zum einen Goto-Anweisungen verschieben und zum anderen Goto-Anweisungen eliminieren. Hierbei wird zu Beginn durch eine Anzahl von Goto-Verschiebe-Transformationen die Programmstruktur so umgeformt, dass jede Goto-Anweisung mit ihrem korrespondierenden Ziel-Label in der gleichen Anweisungssequenz vorhanden ist. Im Anschluss daran werden alle Goto-Anweisungen mit Hilfe von Goto-Eliminations-Transformationen durch äquivalente Kontrollflussstrukturen ersetzt. Zur vereinfachten Beschreibung dieses Ansatzes wird davon ausgegangen, dass jede Goto-Anweisung als konditionale Goto-Anweisung der Form `if (condition) goto Label;` vorliegt[1].

[1] Die unkonditionale Goto-Anweisung kann durch `if (true) goto Label;` repräsentiert werden.

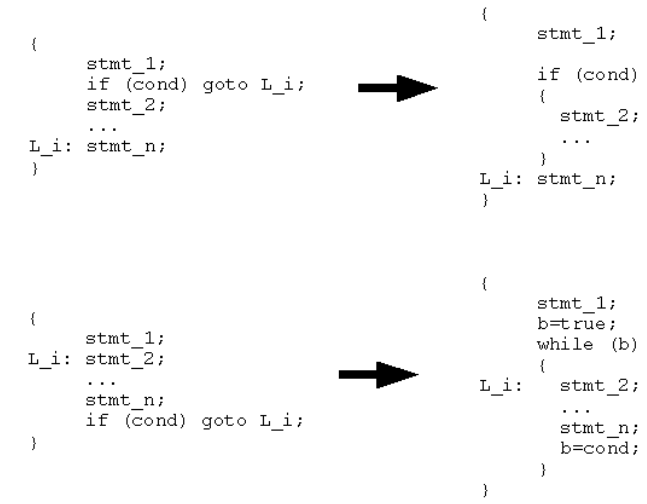

Abbildung 7.22: *Goto-Eliminationstransformationen, die bei der Methode von Erosa und Hendren Anwendung finden.*

Die Goto-Eliminations-Transformationen sind recht einfach, da prinzipiell nur zwei Möglichkeiten existieren, wie Goto-Anweisung und korrespondierendes Ziel-Label in einer Anweisungssequenz zueinander angeordnet werden können. Im Falle, dass die Goto-Anweisung vor dem Ziel-Label steht, wird diese mit Hilfe einer Bedingungsanweisung, die alle zwischen Goto-Anweisung und Ziel-Label vorhandenen Anweisungen als Anweisungsblock enthält, ersetzt. Im Falle, dass diese nach dem korrespondierenden Ziel-Label angeordnet ist durch eine entsprechende Schleifekonstruktion, die alle zwischen Ziel-Label und Goto-Anweisung stehenden Anweisungen beinhaltet. Abbildung 7.22 zeigt die beiden Eliminationstransformationen.

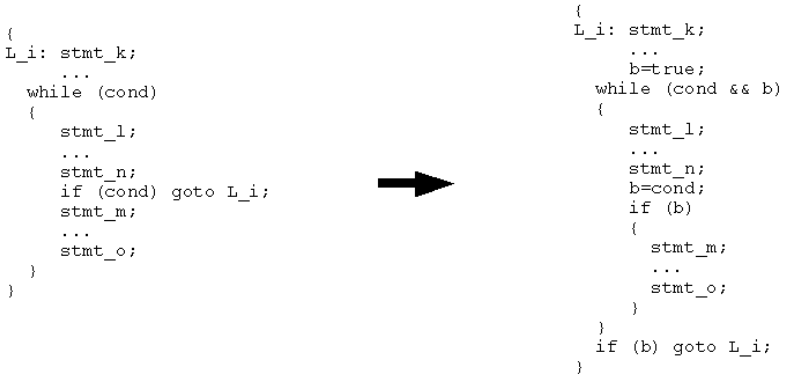

Abbildung 7.23: *Outward-Movement-Transformation, bei der eine Goto-Anweisung aus einer While-Anweisung herausgezogen wird.*

An Goto-Verschiebe-Transformationen lassen sich zwei unterschiedliche Arten unterscheiden: Outward-Movement-Transformationen, die Goto-Anweisungen, die sich auf Ziel-Labels außerhalb des den Goto-

Anweisungsblock umschließenden Anweisungssblock beziehen, in diesen herausziehen und Inward-Movement-Transformationen die Goto-Anweisungen, die sich auf Ziel-Labels innerhalb einer eingebetteten Anweisungssequenz beziehen, in diese hineinziehen. Um den Rahmen dieser Arbeit nicht zu sprengen, soll an dieser Stelle lediglich eine Outward-Movement-Transformation, das Herausziehen einer Goto-Anweisung aus einer While-Anweisung, die beispielsweise im Kontext der Restrukturierung von SEME klassisch While-artigen Schleifen Regionen Anwendung findet, und eine Inward-Movement-Transformation, das Hineinziehen einer Goto-Anweisung in eine If-Anweisung exemplarisch dargestellt werden. An dieser Stelle sei bemerkt, dass das Hineinziehen von Goto-Anweisungen in While-Schleifen im Kontext der Restrukturierung von Regionen nicht berücksichtigt werden muss, da While-artige Regionen mit mehreren Eintrittspfaden nicht existieren. Für die restlichen Transformationen, die zur Beschreibung eines vollständigen Eliminationsansatzes notwendig sind, sei auf die Arbeit von M. Erosa verwiesen.

Das Herausziehen einer Goto-Anweisung aus einer While-Anweisung ist relativ einfach. Es wird eine zusätzliche boolsche Variable b eingeführt, die vor der While-Anweisung mit *true* initialisiert wird und den Wert der Goto-Bedingung speichert sowie eine if-Anweisung, die falls $b \neq true$ ist zur Ausführung der restlichen Anweisungen des While-Blockes führt. Darüberhinaus wird die ursprüngliche While-Bedingung c durch $c\&\&b$ ersetzt und nachfolgend der While-Anweisung eine Goto-Anweisung mit b als Bedingung erzeugt. Abbildung 7.23 verdeutlicht diese Transformation.

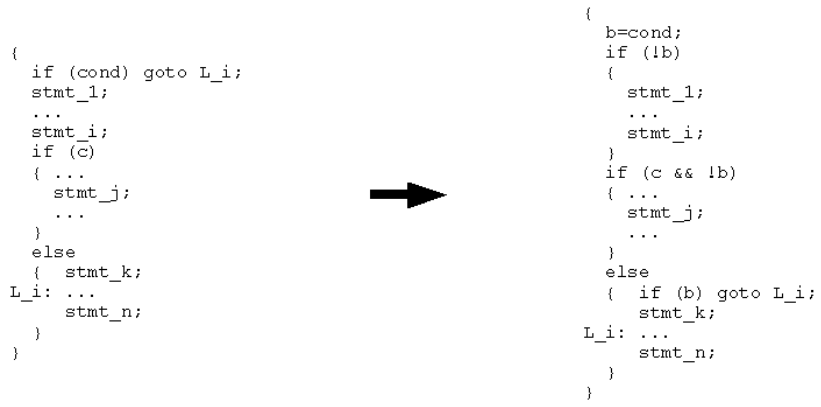

Abbildung 7.24: *Inward-Movement-Transformation, bei der eine Goto-Anweisung in eine If-Anweisung hineingezogen wird.*

Das Hineinziehen einer Goto-Anweisung in eine if-Anweisung gestaltet sich ähnlich. Es wird eine zusätzliche boolsche Variable b eingeführt, die mit der Goto-Bedingung initialisiert wird. Zusätzlich eine If-Anweisung, die falls $b \neq wahr$ ist zur Ausführung aller Anweisungen zwischen Goto-Anweisung und dem Label enthaltenden If-Anweisung führt. Zusätzlich wird die ursprüngliche if-Bedingung c durch $c\&\&!b$ ersetzt und eine Goto-Anweisung mit b als Bedingung als erste Anweisung im if- bzw. else Block erzeugt. Abbildung 7.24 verdeutlicht diese Transformation.

7.7.6 SEME Until-artige Schleifen Regionen

Bei SEME Until-artigen Schleifen Regionen handelt es sich ebenfalls um unstrukturierte Regionen, die ebenso wie die bereits beschriebenen Regionen anhand der einleitend genannten Methoden restrukturiert werden können. An dieser Stelle sei jedoch ein Ansatz charakterisiert, der die bereits im Kontext der Restrukturierung von SESE Until-artigen Schleifen Regionen und SEME klassisch While-artigen

Schleifen Regionen genannten Methoden nutzt.

Hierbei werden SEME Until-artige Schleifen Regionen unter Verwendung einer der in Abschnitt 7.7.3 genannten Methoden, wobei bei Anwendung der Duplikations-Methode zusätzlich alle den Schleifenkörper verlassenden Kanten kopiert werden müssen, in SEME klassisch While-artige Schleifen Regionen transformiert, die anschließend mittels den im vorherigen Abschnitt beschriebenen Methoden in SESE klassisch While-artige Schleifen Regionen restrukturiert werden.

7.7.7 SEME nicht klassisch While-artige Schleifen Regionen

In Analogie hierzu lassen sich ebenfalls SEME nicht klassisch While-artige Schleifen Regionen über den Zwischenschritt der Umwandlung in SEME Until-artige Schleifen Regionen in SESE klassisch While-artige Schleifen Regionen restrukturieren, indem zu Beginn eine beliebige, den Schleifenkörper verlassenden Kante ausgewählt wird und die in Abschnitt 7.7.4 beschriebene Methode mit der ausgewählten Kante als Ausgangskante (ohne Schritt 7) angewendet wird, anhand dessen SEME nicht klassisch While-artige Schleifen Regionen in SEME Until-artige Schleifen Regionen transformiert werden, die anschließend anhand der im vorherigen Abschnitt beschriebene Methode in klassische SESE While-artige Schleifen Regionen überführt werden.

7.7.8 Regionen ohne abnormalen Selection Path

Im Kontext der Abbildung nach BPEL4WS müssen Regionen ohne abnormalen Selection Path ebenfalls geeignet restrukturiert werden, was unter Anwendung der eingangs erwähnten Kontrollflussnormalisierung-Methoden, beispielsweise Erosa u. Henderen, realisiert werden kann. Im Folgenden wird jedoch eine Fluss-Diagramm basierte Methode dargestellt, die die Grundstruktur, unter Ausnutzung spezieller Eigenschaften von Regionen ohne abnormalen Selection Path, bei der Restrukturierung weitestgehend, gegenüber der Anwendung beispielsweise der Methode von Erosa und Hendren, erhält. An dieser Stelle sei bemerkt, dass Regionen ohne abnormalen Selection Path in BPEL4WS grundsätzlich mittels `flow-Aktivitäten` realisierbar sind, indem Kanten als `link`s und Verzweigungsbedingungen als `transitionCondition`s umgesetzt werden, was mit zunehmenden Kantenumfang sehr unübersichtlich und damit hier nicht weiter vertieft wird.

Da Regionen ohne abnormalen Selection Path die Eigenschaft besitzen, dass alle eingehenden Pfade von Verschmelzungsknoten, außer der Region-Verschmelzungsknoten, von einem Verzweigungsknoten, nicht wie bei Regionen mit abnormalen Selection Path von mehreren, ausgehen, besteht die Möglichkeit unter gezielter Einführung von den Verschmelzungsknoten korrespondierenden Verzweigungsknoten solche Regionen anhand normaler Verzweigungs-Regionen, ähnlich zu Multiplen-Merge Regionen (7.7.2), über die daraus resultierende Transformation in Multipe Split Regionen, die anhand der in Abschnitt 7.7.2 beschriebene Methode in eine normale Verzweigungs-Regionen umgewandelt werden, zu restrukturieren. Hierbei wird für jeden Verschmelzungsknoten der ursprüngliche Verzweigungsknoten in zwei Verzweigungsknoten gesplittet, wobei der eine alle ausgehende Pfade, die zu dem Verschmelzungsknoten führen und der andere alle Pfade zu den verschiedenen daraus resultierenden normalen Verzweigungs-Regionen, besitzt. Die daraus resultierenden Multiplen Split Regionen werden anschließend in normale, in BPEL4WS abbildbare Verzweigungs-Regionen restrukturiert. Abbildung 7.25 verdeutlicht diese Restrukturierung.

Anhand der nachfolgenden Schritte wird der Ablauf der Restrukturierung wie folgt charakterisiert:

1. Bestimme die Menge VSK aller Verschmelzungsknoten der Region, ohne den Region-Verschmelzungsknoten k.

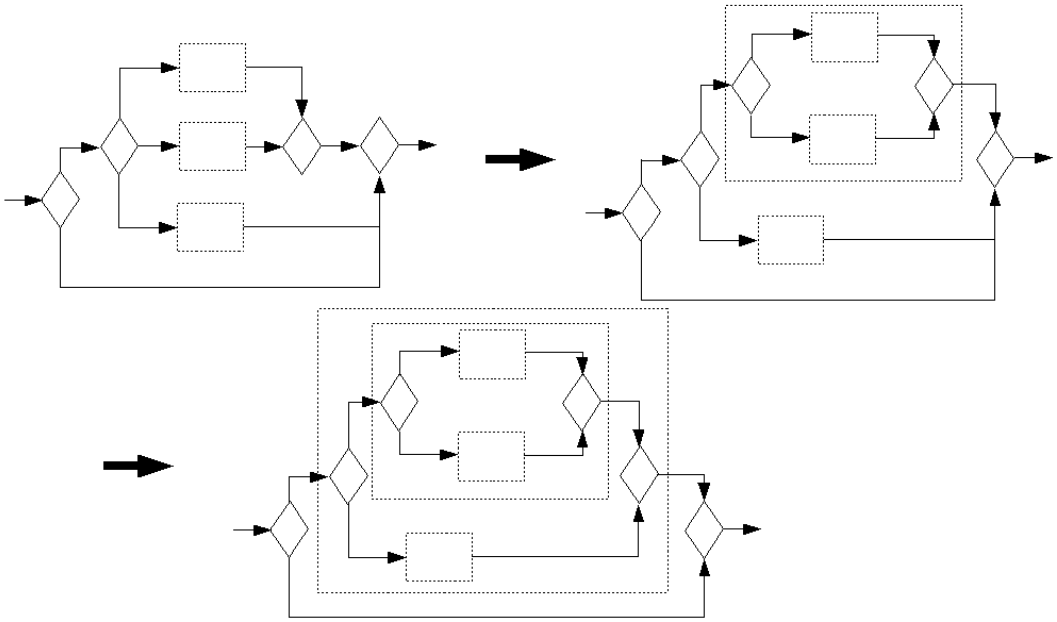

Abbildung 7.25: *Schematische Darstellung der Restrukturierung einer Region ohne abnormal Selection Path über den Schritt der Erzeugung einer Multiplen Split Region.*

2. Bestimme für jeden Verschmelzungsknoten $vsk_i \in VSK$ den zugehörigen Verzweigungsknoten vzk_j, von dem alle in vsk_i mündenden Pfade ausgehen.

3. Wähle einen beliebigen Knoten $vsk_i \in VSK$ und zugehörigen vzk_j aus.

4. Erzeuge einen neuen Verzweigungsknoten vzk_i, der alle bzgl. vsk_i ausgehende Kanten $AK_{vsk_i}(vzk_j)$ von vzk_j besitzt, die bei vzk_j gelöscht werden und der mit vzk_j über eine Kante verbunden ist, deren Bedingung $\bigvee_{a \in AK_{vzk_j}} cond(a)$ ist. Markiere die so neu enstandene Region als Verzweigungs-Region, für $|AK(vzk_i)| = |EK(vsk_i)|$ mit $|AK(vzk_i)| > 2$ als Mehrfach-Verzweigungs-Region und für $|AK(vzk_i)| = 2$ als if-then bzw. if-then-else Region. Speziell für $|AK(vzk_i)| \neq |EK(vsk_i)|$ als bisher unklassifizierte Verzweigungs-Region.

5. Bilde die Menge $VSK = VSK/vsk_i$ und wiederhole Schritt 3, solange $VSK \neq \emptyset$ gilt.

6. Reklassifiziere alle bisher unklassifizierten Verzweigungs-Regionen, indem jeweils der eingehende Verzweigungsknoten $evzk$ betrachtet wird. Für $|AK(evzk) > 2|$ handelt es sich um eine Mehrfach-Verzweigungs-Region, für $|AK(evzk) = 2|$ um eine if-then- bzw. if-then-else Region

7. Restrukturiere die so entstandene Multiple Split Region nach der Methode in Abschnitt 7.7.2.

7.7.9 Abnormale Selection Path Regionen

Wie schon bereits Regionen ohne abnormalen Selection Path gehören ebenfalls Regionen mit abnormalen Selection Path in die Klasse der gegenüber BPEL4WS unstrukturierten Regionen, die ebenso mittels `flow`-Aktivitäten in BPEL4WS realisiert werden können. Prinzipiell lassen sich wiederum alle eingangs erwähnten Normalisierungsmethoden für die Restrukturierung anwenden, was bei den meisten aufgrund der Minimierung der Anzahl an Duplikationen mittels der Einführung zusätzlicher Bedingungen zu sehr gegenüber einem Benutzer unübersichtlichen Kontrollflüssen führt. Dadurch soll

im Folgenden eine Fluss-Diagramm basierte Methode, die zur Restrukturierung Knoten-Duplikation verwendet, dargestellt werden.

Hierbei werden beginnend abnormale Selection Path Regionen in Regionen ohne abnormalen Selection Path restrukturiert, indem alle Selection Path verursachenden Verschmelzungsknoten entfernt werden. Dabei wird für jeden zu entfernenden Verschmelzungsknoten der jeweils für alle außer einer Eingangskante zu duplizierende Sub-Graph bestimmt, der Verschmelzungsknoten nicht mit berücksichtigt, da ansonsten zusätzliche Verzweigungsknoten einzuführen wären, eine beliebige Eingangskante des Verschmelzungsknotens durch die jeweilige Ausgangskante subsummiert und die restlichen Eingangskanten durch jeweils die Kopie der Ausgangskante des Verschmelzungsknoten im Kontext der Sub-Graph-Kopie ersetzt und der Verschmelzungsknoten entfernt. Die daraus resultierenden Regionen, bei denen es sich entweder um Regionen ohne abnormalen Selection Pfad, Multiple Merge oder -Split Regionen handelt, werden nachfolgend entsprechend den in Abschnitt 7.7.8, 7.7.2 beschriebenen Methoden in normale Verzweigungs-Regionen restrukturiert. Abbildung 7.26 verdeutlicht diese Restrukturierung.

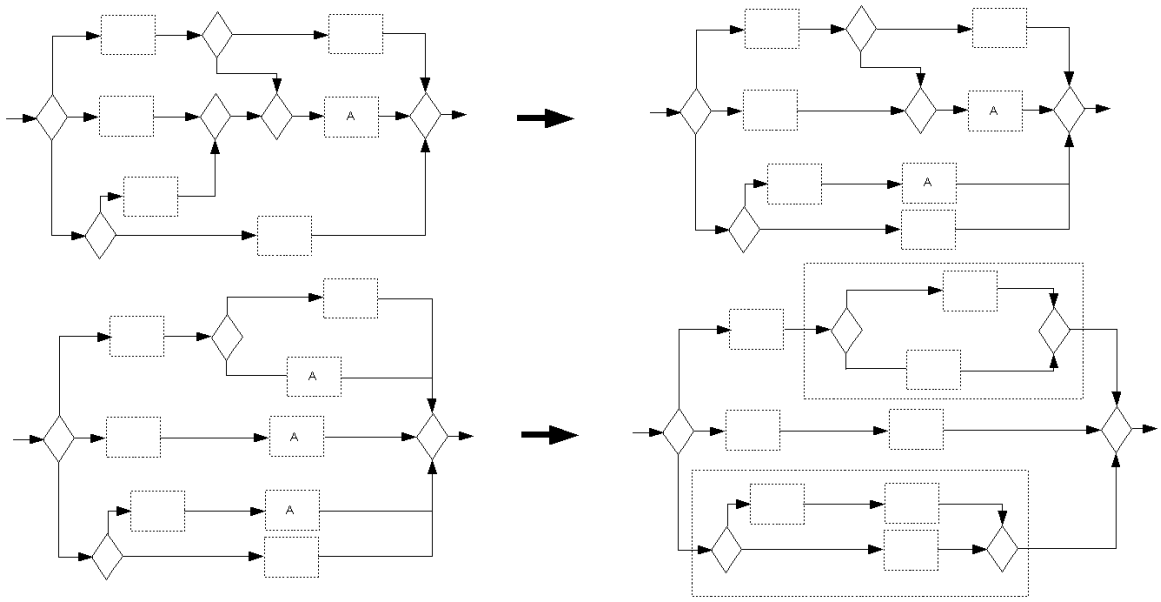

Abbildung 7.26: *Schematische Darstellung der Restrukturierung einer Region mit abnormal Selection Path über den Schritt der Erzeugung einer Multiple Split Region.*

Anhand der nachfolgenden Schritte wird der Ablauf der Restrukturierung wie folgt charakterisiert:

1. Bestimme die Menge $AVSK$ der abnormale Selection Pfade verursachenden Verschmelzungsknoten. Dies kann bereits zum Zeitpunkt der Klassifikation von Regionen realisiert werden.
2. Wähle einen Verschmelzungsknoten $avsk_i \in AVSK$ aus.
3. Erzeuge eine Kopie von $avsk_i \in AVSK$, die lediglich ein Eingabe- und Ausgabe-Pin besitzt, welches mit einer Eingangskante $ek_j \in EK(avsk_i)$ und einer Kopie der Ausgangskante ak verbunden wird. Bilde $EK(avsk_i)/ek_j$.
4. Bestimme den Sub-Graphen SG_{avsk_i}, der alle Knoten und Kanten der möglichen Pfade von $avsk_i$, die in einem gemeinsamen Verschmelzungsknoten zusammenlaufen, repräsentiert. Hier-

bei werden alle Pfade vom Verschmelzungsknoten zum Region-Verschmelzungsknoten traversiert und der erste in allen Pfaden vorhande Verschmelzungsknoten, der den gemeinsamen Verschmelzungsknoten repräsentiert, ermittelt, sowie die einzelnen Kantenmengen von nicht traversierten Verschmelzungsknoten-Kanten bereinigt. Der Sub-Graph wird ebenfalls von der in 2 erzeugten Kopie der Ausgangskante bereinigt.

5. Für jede Eingangskante $ek_k \in EK(avsk_i)$ kopiere den Sub-Graph SG_{avsk_i} und setze den Zielknoten der Eingangskante ek_k durch die Kopie von $avsk_i$ im Sub-Graphen.

6. Entferne $avsk_i$ und dessen Ausgangskante, bilde $AVSK/avsk_i$ und gehe zu Schritt 2 falls $AVSK \neq \emptyset$.

7. Re-klassifiziere die so entstande Region. Handelt es sich um eine Multiple Split bzw. -Merge Region wende die in Abschnitt 7.7.2 beschriebene Methode an, bei einer Region ohne abnormalen Selection Path, die in Abschnitt 7.7.8.

7.7.10 Multi Loop Regionen ohne Überlappung

In die Klasse der unstrukturierten Regionen werden ebenfalls Multi Loop Regionen ohne Überlappung eingeordnet, da es sich grundsätzlich um Regionen handelt, deren Schleifen mehrere Austrittspfade besitzen, die, sonst wären sie nicht in einer SESE-Region erfasst, durch mehrere gemeinsame Verschmelzungsknoten zusammengeführt werden[2]. Wie bereits die vorherig betrachteten, unstrukturierten Regionen lassen sich ebenfalls Multi Loop Regionen mittels den eingangs benannten Normalisierungsmethoden restrukturieren, wobei an dieser Stelle, aufgrund der nicht Überlappung der einzelnen Schleifen, eine Methode, die zu Beginn die Einzelschleifen restrukturiert, angewendet werden kann, die diese spezielle Eigenschaft gegenüber anderen Methoden wie die von Erosa oder Zang, die entweder die komplette Region oder mehrere Schleifen umfassenden Sub-Graphen restrukturieren, gezielt berücksichtig.

Hierbei werden zu Beginn durch eine lokale, nicht über Regionen stattfindende Restrukturierung alle SEME- in SESE-Schleifen umgewandelt, wodurch neue SESE-Regionen entstehen, die als atomare Knoten weiter behandelt werden. Dadurch werden Multi Loop Regionen in Regionen mit abnormalen Selection Path überführt, die anschließend unter Verwendung der bereits beschriebenen Methoden in klassische Verzweigungs-Region restrukturieren werden. Abbildung 7.27 zeigt schematisch die Restrukturierung.

Anhand der nachfolgenden Schritte wird der Ablauf der Restrukturierung wie folgt charakterisiert:

1. Bestimme die Menge S der Schleifenquell-Zielknotenpaare (sqk, szk), die jeweils alle Schleifen innerhalb der Region charakterisieren. Dies kann bereits im Kontext der Klassifikation im Schritt 5i realisiert werden.

2. Wähle eine beliebige Schleife $s \in S$ aus.

3. Bestimme den Typ der Schleife s. Falls $|AK(sqk)| = 2$, so handelt es sich um eine SEME Until Schleife. Falls $|AK(sqk)| = 1$, überprüfe ob der dem Schleifenzeilknoten folgende Knoten ein Verzweigungsknoten ist, der eine den Schleifenkörper verlassende Kante besitzt. Wenn ja, handelt es sich um eine SEME klassische While Schleife, wenn nein, um eine SEME nicht klassische While Schleife.

[2] gemeinsam bedeutet in diesem Kontext, dass durch einen Verzweigungsknoten beispielsweise ein Austrittspfad der Schleife A und der Schleife B zusammengeführt werden.

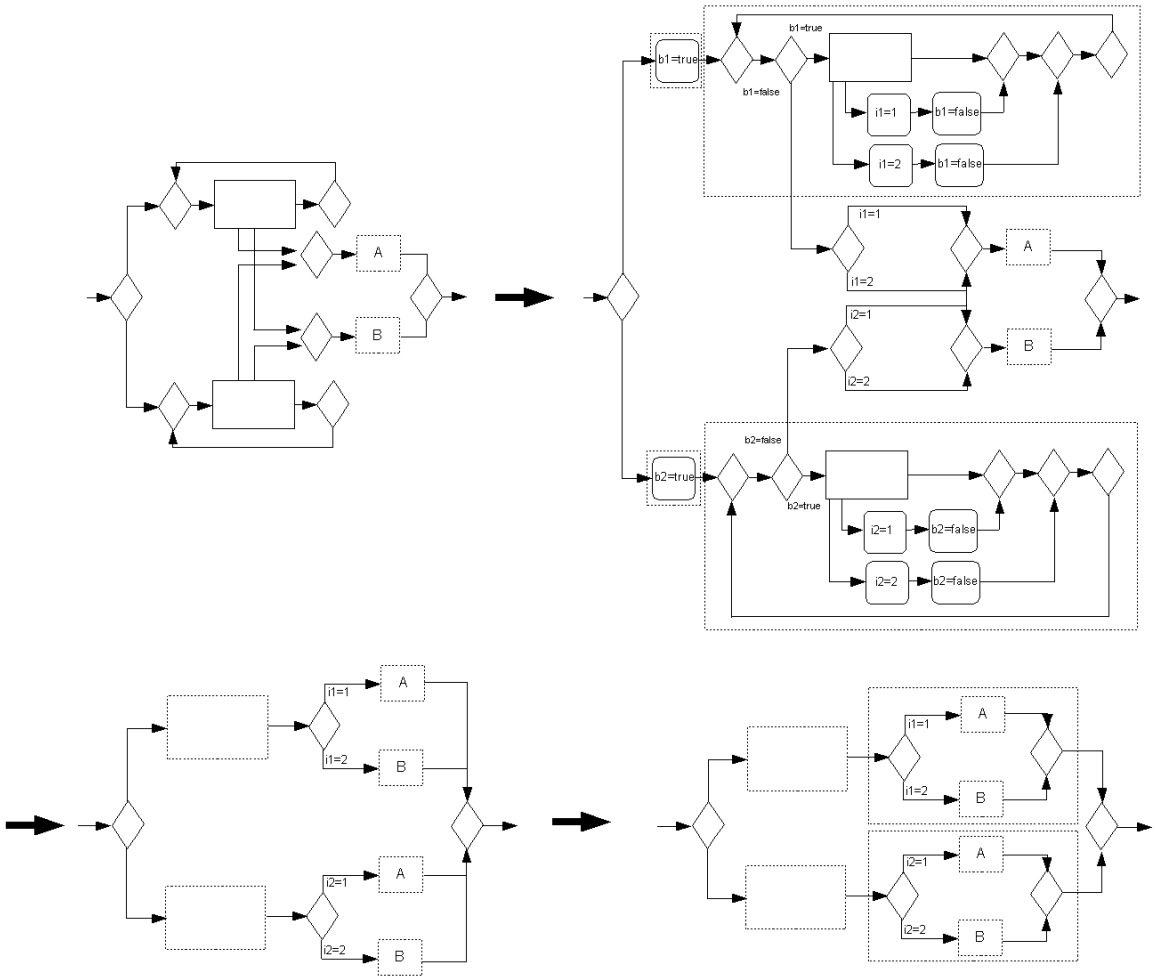

Abbildung 7.27: *Schematische Darstellung der Restrukturierung einer Multi Loop Region mit sich nicht überlappenden Schleifen über den Schritt der Erzeugung und Restrukturierung einer abnormalen Selection Path- und Multiplen Split Region.*

4. Führe eine lokale Restrukturierung der Schleifen durch, indem SEME Until Schleifen mittels der in Abschnitt 7.7.6 beschriebenen Methode, die ebenfalls auf nicht SESE-Regionen anwendbar ist und SEME nicht klassisch While Schleifen anhand der in Abschnitt 7.7.7 beschriebene Methode in SEME klassische While Schleifen restrukturiert werden.

5. Restrukturiere die in SEME klassische While Schleifen umgewandelten Schleifen anhand der Methode von Williams in klassische SESE While Schleifen.

6. Erzeuge für die so entstandenen SESE klassischen While Schleifen jeweils eine SESE klassisch While-artige Schleifen Region.

7. Bilde S/s und gehe zu Schritt 2, falls $S \neq \emptyset$.

8. Wende auf die in eine Region mit abnormalen Selection Path transformierte Region die in Abschnitt 7.7.9 beschriebene Methode zur Restrukturierung in eine normale Verzweigungs-Region an.

7.7.11 Multi Loop Regionen mit partieller Überlappung

Neben Multi Loop Regionen ohne Überlappung existieren Multi Loop Regionen mit partieller Überlappung, die neben nicht-überlappenden Schleifen auch überlappende beinhalten und ebenfalls in die Klasse der unstrukturierten Regionen einzuordnen sind.

Bezüglich der Restrukturierung können die eingangs erwähnten Normalisierungsmethoden angewendet werden. Es besteht aber die Möglichkeit über den Schritt der Restrukturierung der nicht überlappendenen in klassische SESE-Schleifen mittels der in 7.7.10 beschriebenen Methode Multi Loop Regionen mit partieller Überlappung in Multiple Loop Regionen mit vollständiger Überlappung umzuwandeln, die anschließend mit den dafür spezifizierten Methoden restrukturiert werden.

7.7.12 Multi Loop Regionen mit vollständiger Überlappung

Wie Multi Loop Regionen mit partieller Überlappung gehören ebenfalls Multi Loop Regionen mit vollständiger Überlappung in die Klasse der unstrukturierten Regionen, für die, aufgrund ihrer strukturellen Komplexität, an dieser Stelle, gegenüber den in den vorherigen Abschnitt dargestellten, spezifisch auf die Struktur ausgerichteten Restrukturierungs-Methoden, keine spezielle Methode spezifiziert werden kann, so dass die Restrukturierung an dieser Stelle entweder durch allgemeine, automatisierte Methoden wie [15, 3, 50, 27] oder beispielsweise einen Prozess-Designer durchzuführen ist.

7.8 Beispiel

Im Folgenden soll anhand des in Abbildung 7.28 schematisch dargestellten BPDM Prozesses der Ablauf der Abbildung des Kontrollflusses unter Verwendung des hier charakterisierten Ansatzes in BPEL4WS demonstriert werden.

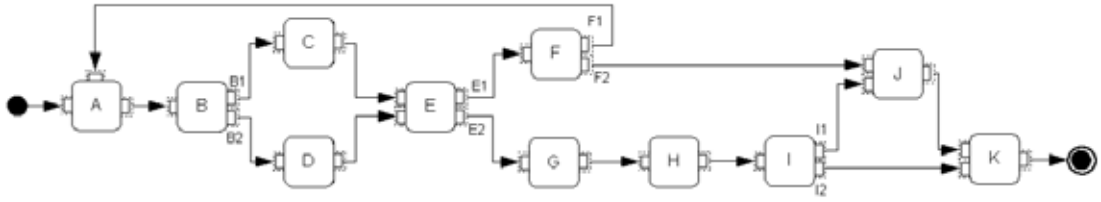

Abbildung 7.28: *BPDM Beispiel Prozess.*

Im ersten Schritt wird der Kontrollfluss des BPDM Prozesses in Form eines äquivalenten Prozessflussgraphen repräsentiert, Abbildung 7.29. Die sich anschließende einleitende Tranformation nach Abschnitt 7.4 ist als solches nicht notwendig, da keine kombinierten Verschmelzungs-/Verzweigungsknoten vorhanden sind.

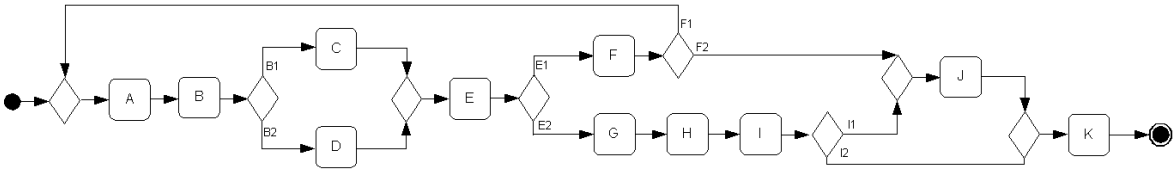

Abbildung 7.29: *Repräsentation des Kontrollflusses des BPDM Prozesses als Prozessflussgraph.*

Im Anschluss daran werden mittels der in 7.5 charakterisierten Methode alle kanonischen SESE-Regionen identifiziert, sowie davon ausgehend der PST abgeleitet. Die identifizierten SESE-Regionen und der daraus abgeleitete PST sind in Abbildung 7.30 dargestellt.

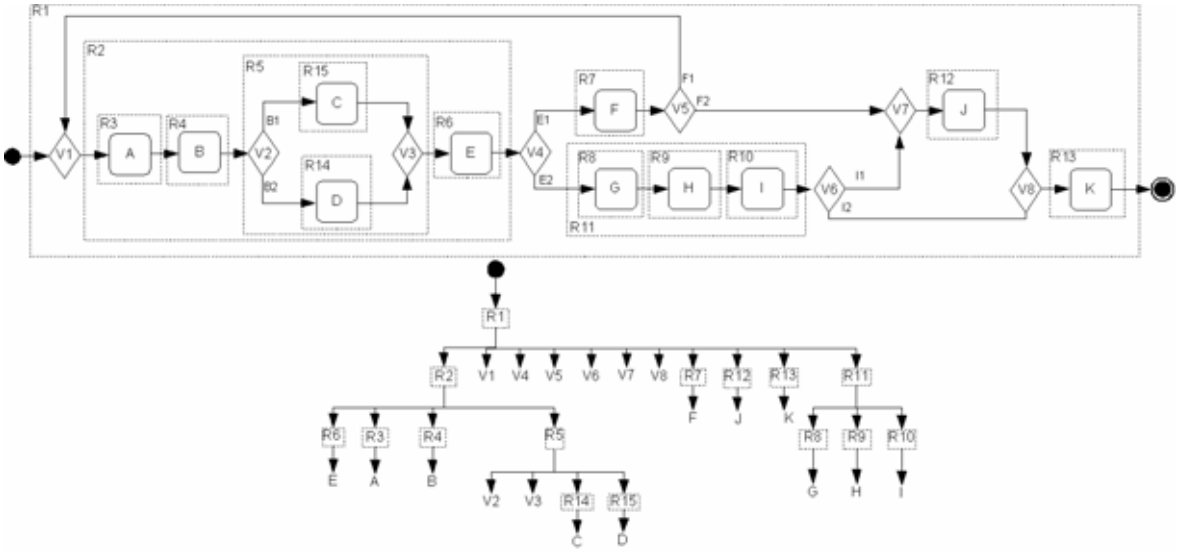

Abbildung 7.30: *Prozess mit identifizierten SESE-Regionen sowie entsprechendem PST.*

Anschließend werden alle SESE-Regionen anhand des in Abschnitt 7.6 beschriebenen Klassifikationsalgorithmus typisiert. Daraus resultiert, dass es sich bei $R1$ um eine SEME Until-artige Schleifen Region, bei $R2$ und $R11$ um Sequenz Regionen, bei $R5$ um eine If-then-else Region und bei $R3$, $R4$, $R15$, $R14$, $R6$, $R7$, $R8$, $R9$, $R10$, $R12$ und $R13$ um Basic-Entity Regionen handelt.

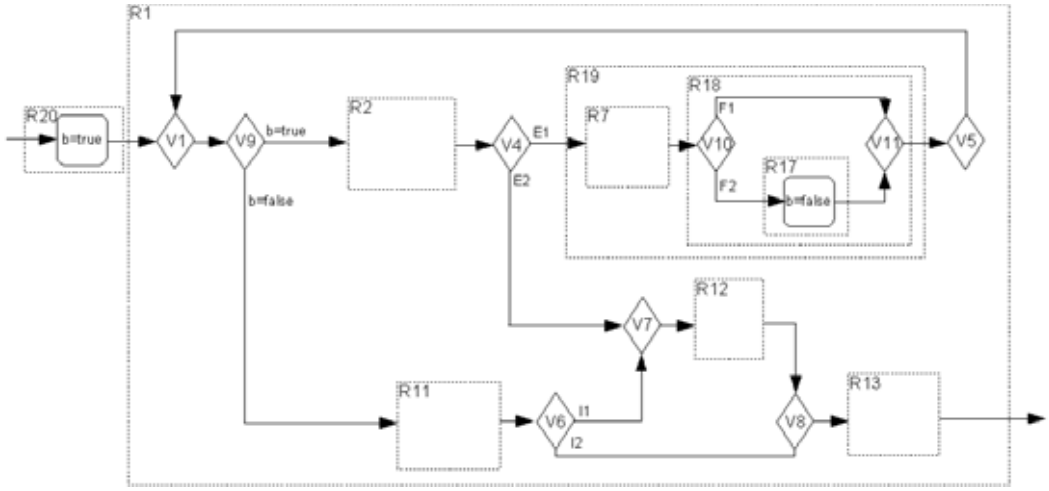

Abbildung 7.31: *Als SEME klassisch While-artige Schleifen Region restrukturierte Region $R1$.*

Da lediglich SEME Until-artige Schleifen Regionen gegenüber BPEL4WS als unstrukturierte Regionen charakterisiert sind, ist lediglich die Region $R1$ für die Abbildung zu restrukturieren. Hierzu wird die SEME Until-artige Schleifen Region zu Beginn anhand der in Abschnitt 7.7.6 beschriebenen Methode über die Einführung einer zusätzlichen Variable in eine SEME klassisch While-artige Schleifen

Region, die in Abbildung 7.31 dargestellt ist, restrukturiert. Dabei entstehen neue Regionen $R18$, $R19$, $R17$ und $R20$, die als if-then-, Sequenz- und Basic-Entity Regionen typisiert sind und bzgl. der Restrukturierung nicht weiter betrachtet werden müssen.

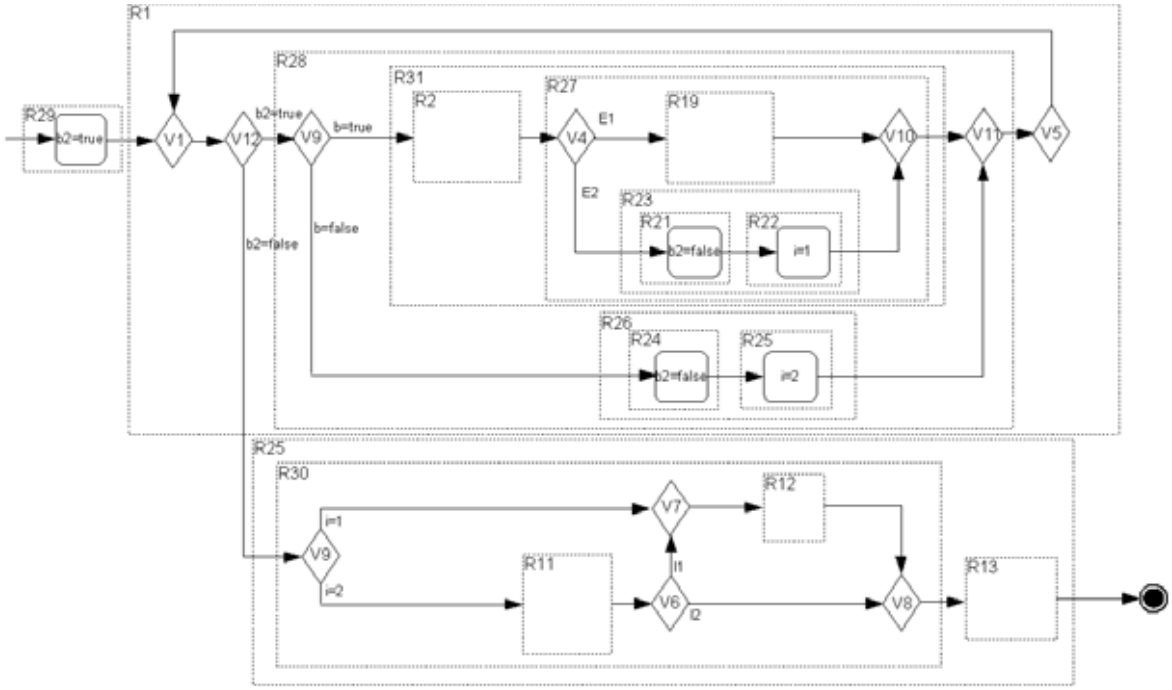

Abbildung 7.32: *Als SESE klassisch While-artige Schleifen Region restrukturierte Region* $R1$.

Die SEME klassisch While-artige Schleifen Region $R1$ wird anschließend anhand der in Abschnitt 7.7.5 beschriebene Methode von Williams und Ossher in eine SESE klassisch While-artige Schleifen Region, Abbildung 7.32, weiter restrukturiert. Hierbei entstehen neue Regionen $R21$, $R22$, $R24$, $R25$ und $R29$, die als Basic-Entity Regionen, $R26$, $R23$, $R25$ und $R31$ die als Sequenz-Regionen, $R27$ und $R28$, die als if-then-else Regionen und $R30$, die als Region mit abnormal Selection Path typisiert sind. Da es sich mit Ausnahme von $R30$ um gegenüber BPEL4WS strukturierte Regionen handelt, brauchen diese im Folgenden nicht weiter restrukturiert werden. Lediglich $R30$ wird anhand der in Abschnitt 7.7.9 charakterisierten Methode in eine multiple Split Region, Abbildung 7.33, weiter restrukturiert.

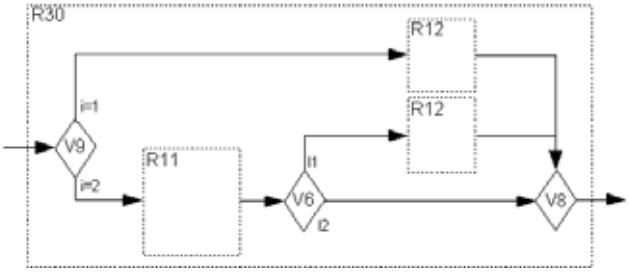

Abbildung 7.33: *Als multiple Split Region restrukturierte Region* $R30$.

Diese wird abschließend mittels der in Abschnitt 7.7.2 beschriebenen Methode in eine if-then-else Region, Abbildung 7.34, restrukturiert, wodurch eine neue if-then Region, $R32$, sowie eine Sequenz-Region, $R33$, entstehen.

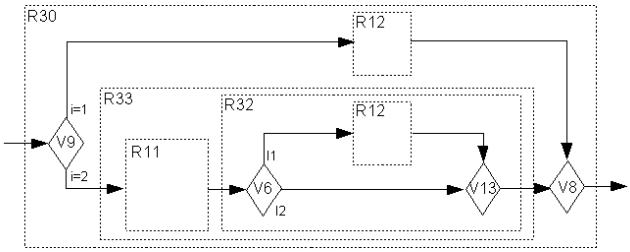

Abbildung 7.34: *Als if-then-else Region restrukturierte Region* $R30$.

Den daraus resultierenden, restrukturierten und direkt in BPEL4WS abbildbaren BPDM Prozess zeigt zur Vollständigkeit Abbildung 7.35.

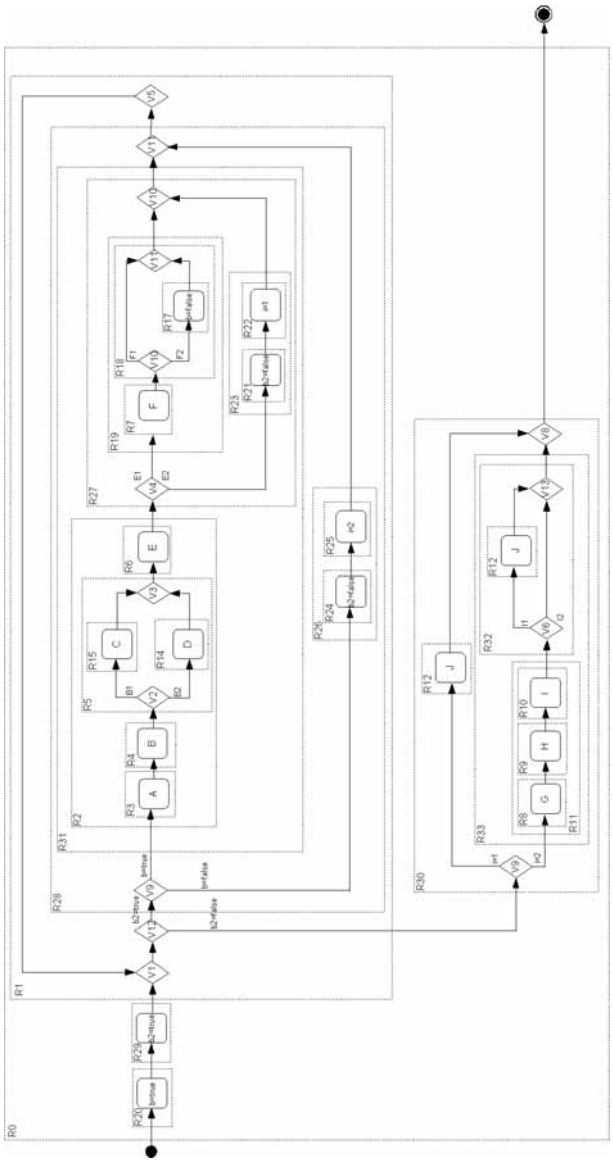

Abbildung 7.35: *Restrukturierter, in BPEL4WS abbildbarer, Prozess.*

Abschließend wird der so restrukturierte Kontrollfluss des BPDM Prozesses in BPEL4WS über die äquivalente Umsetzung der einzelnen Regionen mit BPEL4WS Kontrollfluss-Konstrukten, Abschnitt 7.7 und 5.1.4, abgebildet. Das nachfolgende Listing zeigt den daraus resultierenden BPEL4WS Code für den Beispiel BPDM Prozess.

```
<process ...>
    ...
    <variables>
        <variable name="b"  type="xsd:bool">
        <variable name="b2" type="xsd:bool">
        <variable name="i"  type="xsd:integer">
    <variables>
    ...

    <sequence name="R0">
        <assign name="R20">
            <copy>
                <from expression="true"/>
                <to variable="b"/>
            </copy>
        </assign>
        <assign name="R29">
            <copy>
                <from expression="true"/>
                <to variable="b2"/>
            </copy>
        </assign>
        <while condition="b2=true" name="R1">
            <switch name="R28">
                <case condition="b=true">
                    <sequence name="R31">
                        <sequence="R2">
                            <invoke A name="R3" .../>
                            <invoke B name="R4" .../>
                            <switch name="R5">
                                <case condition="B1">
                                    <invoke C name="R15" .../>
                                </case>
                                <case condition="B2">
                                    <invoke D name="R14" .../>
                                </case>
                            </switch>
                            <invoke E name="R6" .../>
                        </sequence>
                        <switch name="R27">
                            <case condition="E1">
                                <sequence name="R19">
                                    <invoke F name="R7" .../>
                                    <switch name="R18">
                                        <case condition="F2">
                                            <assign name="R17">
                                                <copy>
                                                    <from expression="false"/>
                                                    <to variable="b"/>
```

```
                                        </copy>
                                    </assign>
                                </case>
                            </switch>
                        </sequence>
                    </case>
                    <case condition="E2">
                        <sequence name="R23">
                            <assign name="R21">
                                <copy>
                                    <from expression="false"/>
                                    <to variable="b2"/>
                                </copy>
                            </assign>
                            <assign name="R22">
                                <copy>
                                    <from expression="1"/>
                                    <to variable="i"/>
                                </copy>
                            </assign>
                        </sequence>
                    </case>
                </switch>
            </sequence>
        </case>
        <case condition="b=false">
            <sequence name="R26">
                <assign name="R24">
                    <copy>
                        <from expression="false"/>
                       <to variable="b2"/>
                    </copy>
                </assign>
                <assign name="R25">
                    <copy>
                        <from expression="2"/>
                        <to variable="i"/>
                    </copy>
                </assign>
            </sequence>
        </case>
    </switch>
</while>
<switch name="R30">
    <case condition="i=1">
        <invoke J name="R12" .../>
    </case>
    <case condition="i=2">
        <sequence name="R33">
            <invoke G name="R8" .../>
            <invoke H name="R9" .../>
            <invoke I name="R10" .../>
            <switch name="R32">
                <case condition="I1">
                    <invoke J name="R12" .../>
```

```
                    </case>
                </switch>
            </sequence>
        </case>
    </switch>
</sequence>
</process>
```

Hierbei ist der daraus resultierende BPEL4WS Code noch unvollständig, da die detaillierte Abbildung beispielsweise von den im BPDM Prozess enthaltenen Steps nach 5.1 nur angedeutet wurde.

7.9 Ausblick

Der in diesem Kapitel vorgestellte allgemeine Ansatz zur Abbildung des Kontrollflusses von Prozessen (darunter auch BPDM Prozessen), äquivalent zu Prozessflussgraphen, in BPEL4WS stellt gegenüber ganzheitlichen Ansätzen aufgrund seiner divide-and-conquer Strategie eine flexible Methode dar, die neben der Abbildung von Kontrollflüssen beispielsweise in Modellierungstools zur Unterstützung des Entwurfs und der Analyse von Prozessmodellen eingesetzt werden kann, indem zum Beispiel bereits während der Design-Zeit dem Prozess-Designer gegenüber der Zielsprache unstrukturierte Regionen visuell charakterisiert oder nicht notwendige Details des Prozess-Modells beim Entwurf ausgeblendet bzw. bereits modellierte Prozesse in verschiedenen Detailgraden angezeigt werden können.

Da aufgrund der Komplexität und des Umfangs dieses Ansatzes in diesem Kapitel lediglich Prozesse, deren Kontrollfluss denen von Prozessflussgraphen entspricht, betrachtet wurden, muss zusätzlich für die vollständige Spezifikation das Klassifikationsschema und der Klassifikationsalgorithmus des bisherigen Ansatzes auf BPDM Prozesse mit Parallelität ausgedehnt, sowie für unstrukturierte Regionen mit Parallelität geeignete Restrukturierungsmethoden spezifiziert werden.

Kapitel 8

Zusammenfassung

In der hier vorliegenden Arbeit wurde die Abbildung von Geschäftsprozessen, spezifiziert im Business Process Definition Metamodel, in die Business Process Execution Language for Web Services speziell die dabei auftretenden Komplikationen und Limitationen, untersucht, sowie damit verbunden für einzelne Problemfelder mögliche Lösungsansätze entwickelt. Ausgangspunkt für diese Untersuchung bildete hierbei die Business Process Execution Language for Web Services in der Version 1.1 und das durch [16] und [21] vorläufig charakerisierte Business Process Definition Metamodel.

Im ersten Abschnitt dieses Kapitels werden die wesentlichen Ergebnisse dieser Arbeit zusammengefasst, sowie im zweiten Abschnitt abschließend ein Ausblick auf mögliche Weiterführungen gegeben.

8.1 Ergebnisse

Im Kontext dieser Arbeit wurde eine mögliche Abbildung von BPDM Prozessen in BPEL4WS spezifiziert, mit der gezeigt wurde, dass viele der von BPDM zur Modellierung gebotenen Konzepte in äquivalenter Form ebenfalls von BPEL4WS unterstützt werden und durch zusätzliche von Seiten des Prozess-Designers spezifizierte Plattform-Informationen in BPEL4WS abbildbar sind, beispielsweise BPDM Prozess-Schnittstellen als BPEL4WS Prozess zugeordnete Service Operationen in Form von WSDL, Receive- oder AcceptMessageTask Tasks als äquivalente `receive`- bzw. `reply`-Aktivitäten oder BPDM Exceptions als äquivalente BPEL4WS Faulthandler. Das aber aufgrund der Modellierungsmöglichkeiten sowie der nicht speziell auf ausführbare Geschäftsprozesse beschränkten Ausrichtung der Spezifikation von Geschäftsprozessen Konzepte und Sprachelemente im BPDM existieren, die aufgrund von Limitationen von BPEL4WS, mitverursacht durch die Plattformabhängigkeit und die Ausrichtung auf die Spezifiaktion ausführbarer Geschäftsprozesse, zu Komplikationen und Schwierigkeiten bei der Abbildung führen. Solche Limitationen sind:

- die fehlende Unterstützung des Subprozess-Konzepts.
- die fehlende Unterstützung zur Spezifikation kollaborativer Prozesse.
- das gegenüber BPDM eingeschränkte und bisher nicht weiter spezifizierte Konzept abstrakter BPEL4WS Prozesse.
- die fehlende Unterstützung zur Modellierung manuell auszuführender Arbeitsschritte sowie die damit verbundene fehlende Unterstützung zur Ressourcenmodellierung.
- die gegenüber BPDM eingeschränkten Möglichkeiten zur Kontroll- und Datenflussmodellierung.
- die fehlende Unterstützung des Konzepts prozessübergreifender Daten in Form von Repositories.

- die gegenüber BPDM eingeschränkten Möglichkeiten zur Modellierung von Prozess-Instanz-Nachrichten Korrelationen.

- sowie das Fehlen einer zum ForEach-Task äquivalenten BPEL4WS Aktivität.

Trotz dieser Limitationen und den damit verbundenen Komplikationen wurde auf Basis des von BPEL4WS angebotenen Extensions-Mechanismus sowie auf Basis äquivalenter, mit den von BPEL4WS zur Verfügung gestellten Sprachkonstrukten und unter Einbeziehung bisher vorgeschlagener Spracherweiterungen [12], semantischer Realisierungen für verschiedene Abbildungsprobleme Lösungsvorschläge spezifiziert, die die Abbildbarkeit von BPDM- in BPEL4WS-Prozesse erhöhen. Beispielsweise wurde gezeigt, dass ForEach-Tasks mittels `while`-Aktivitäten bzw. sparaten BPEL4WS Prozessen oder durch eine im Kontext von [12] zur Erweiterung von BPEL4WS vorgeschlagenen `forEach`-Aktivität in BPEL4WS abbildbar sind, dass Repositories mittels externen Datenbanken, die über Web Service Schnittstellen zum Zugriff auf die zwischen Prozessen auszutauschenden Informationen verfügen, zwar nicht direkt, jedoch über den Umweg von Web Service Aufrufen abbildbar sind, dass der in BPEL4WS nicht repräsentierbare Kontrollfluss von BPDM Prozessen über den Schritt der äquivalenten Restrukturierung mittels im Kontext der Compiler-Optimierung angewendeten Kontrollfluss-Normalisierungmethoden anhand einer den BPDM Prozess in jeweils voneinander unabhängige Regionen zerlegenden divide-and-conquer Strategie in BPEL4WS überführbar ist oder dass die in BPDM mittels kollaborativen Prozessen spezifizierten Geschäftsprotokolle zwischen Rollen zwar nicht als solches, jedoch über die Abbildung in jeweils einen abstrakten Prozess für jede beteiligte Rolle in BPEL4WS mit der Einschränkung des Verlusts der Neutralität abbildbar sind. Darüber hinaus wurde festgestellt, dass neben Limitationen, die durch äquivalente Transformation beseitigt werden können, andere, wie die fehlende Unterstützung zur Abbildung manueller Tasks oder Subprozesse und die fehlende Unterstützung abstrakter Prozesse zur Spezifikation der jeweiligen sie realisierenden Prozesse, ausschließlich durch nicht standardisierte, propritäre Erweiterungen auf Seiten von BPEL4WS behoben werden können, was speziell bei Verwendung dieser die möglichen Ausführungsumgebungen auf die einschränkt, die diese Erweiterungen unterstützen; wodurch es in solchen Fällen sinnvoll erscheint, deren Verwendung von beispielsweise einem Prozess-Designer abhängig zu machen.

Ein weiteres Ergebnis dieser Arbeit ist, dass Sprachkonstrukte wie beispielsweise Korrelationsprädikate existieren, die nicht vollständig bzw. nur sehr eingeschränkt in BPEL4WS abbildbar sind bzw. für die es, wie beispielsweise für die Konstrukte zur Ressourcenmodellierung, keinen Sinn macht, aufgrund der Ausrichtung von BPEL4WS, diese in BPEL4WS repräsentieren zu wollen bzw. entsprechende dafür mögliche Lösungsansätze zu spezifizieren.

8.2 Ausblick

Da der Prozess der Standardisierung des Business Process Definiton Metamodels bisher noch nicht vollständig abgeschlossen ist kann, diese Arbeit nur als vorläufig auf dem Weg der Spezifikation einer Abbildung von BPDM in BPEL4WS angesehen werden. Damit bleibt als weitere Arbeit die möglicherweise neuen bzw. geänderten, im endgültigen Standard enthaltenen, Sprachkonstrukte und Konzepte bezüglich der Abbildbarkeit nach BPEL4WS zu analysieren, die im Kontext dieser Arbeit aufgezeigten Problemfelder auf ihre Gültigkeit zu überprüfen bzw. für neu entdeckte Problemfelder Lösungsansätze zu spezifizieren sowie die bisherige charakterisierte Abbildung von BPDM in BPEL4WS entsprechend zu aktualisieren und aufgrund der lediglich für diese Arbeit groben Charakterisierung der Abbildung für die eigentliche Identifikation möglicher Limitationen und Komplikationen für eine konkrete Realisierung zu detaillieren. Vor allem aber den in Kapitel 7 charakterisierten Ansatz zur Kontrollfluss-Abbildung hinsichtlicher Parallelität, unabhängig vom BPDM, zu erweitern.

Desweiteren müssen Ansätze und Methoden zur Abbildung von Modellen, speziell Geschäftsprozessmodelle, die im Kontext der MDA mit Hilfe von Metamodellen spezifiziert sind, auf ihre Eignung zur Abbildung von Geschäftsprozessen untersucht werden. Speziell, ob sich beispielsweise der MOF Query/View/Transformation (QVT) Standard hierfür eignet oder das im Kontext des Eclipse Projekt sich in der Entwicklung befindliche Modell Transformation Framework, welches die Abbildung nicht direkt, sondern über die Charakterisierung von Relationen zwischen Quell- und Zielkonstrukt der jeweiligen Metamodelle, spezifiziert und damit gleichzeitig Validierungsaussagen über existierenden Modelle zulässt sowie gegenüber herkömmlichen Verfahren den Vorteile der Trennung von Analyse und Transformation besitzt. Damit verbunden bleibt zu untersuchen, wie der Relationsansatz für den hier spezifizierten Ansatz zur Kontrollflussabbildung, speziell der Klassifikation und Transformation des PFG, eingesetzt werden kann.

Literaturverzeichnis

[1] Aho, A.; Sethi, R.; Ullman, J.; (1986): *Compilers-Principles, Techniques and Tools*; Addison-Wesley

[2] Allen, R.; (2001): *Workflow: An Introduction.*; Fischer, L. (Ed.): Workflow Handbook 2001. Workflow Management Coalition; Future Strategies; Lighthouse Point (FL); 2001.

[3] Armaguellat, Z.; (1992): *A Control Flow Normalization Algorithm and Its Complexity*; Center of Supercomputing Research and Development University of Illinoise; Software Engineering Vol.18 Nr. 3: 237-251

[4] Atkinson, B.; Bellwood, T.; Cahuzac, M.; (2003): *UDDI Version 3.0.1 UDDI Spec Technical Committee Specification, Dated 20031014*; Zugriff 26.10.04; `http://uddi.org/pubs/uddi-v3.0.1-20031014.htm/`

[5] Belaunde, M.; Burt, C.; Casanave, C.; (2003): *MDA Guide V1.0.1*; OMG Document omg/2003-06-01

[6] Bock, C.; (2003): *UML 2 Action and Activity Models*; Journal of Object Technology; vol. 2, no. 4, Juli-August 2003, Seite 43-53; `http://www.jot.fm/issues/issue_2003_07/column3/`

[7] Bock, .C; (2003): *UML 2 Activity and Action Models, Part 2: Actions*; Journal of Object Technology; vol. 2, no. 5, September-Oktober 2003, Seite 41-56; `http://www.jot.fm/issues/issue_2003_09/column4/`

[8] Bock, C.; (2003): *UML 2 Activity and Action Models, Part 3: Control Nodes*; Journal of Object Technology; vol. 2, no. 6, November-Dezember 2003, Seite 7-23; `http://www.jot.fm/issues/issue_2003_11/column1/`

[9] Bock, C.; (2004): *UML 2 Activity and Action Models Part 4*; Journal of Object Technology; vol. 3, no. 1, Januar-Februar 2004, Seite 27-41; `http://www.jot.fm/issues/issue_2003_07/column3/`

[10] Böhm, C.; Jacopini, G.; (1966): *Flow diagrams, Turing machines and languages with only two formation rules*; Communication of the ACM; Vol. 9 Nr. 5: 366-371; Mai 1966

[11] Born, M.; Holz, E.; Kath, O.; (2003): *Softwareentwicklung mit UML2; Die neuen Entwurfstechniken UML2, MOF 2 und MDA*; Addison-Wesley Verlag

[12] *BPEL4WS Issue Liste*; `http://www.choreology.com/external/WS_BPEL_issues_list.html`

[13] *Buisness Process Execution Language for Web Services*; (2003); Version 1.1; Mai 2003; `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/`

[14] Christensen, E.; Curbera, F.; Meredith, G.; (2001): *Web Service Description Language (WSDL) 1.1*; W3C; Zugriff 03.07.04, Version vom 15.03.2001; `http://www.w3.org/TR/wsdl/`

[15] Erosa, A.M.; Hendren, L. J.; (1993): *Taming Control Flow: A Structured Approach to Eliminating Goto Statements*; ACAPS Technical Memo 76, McGill Univesity, School of Computer Science; 29. September 1993;

[16] Frank, J.H.; Gardner, T.A.; Johnston,S.K.; (2004): *Buisness Process Definition Metamodel - Concepts and Overview*; Version 2004-04-08

[17] Goland, Y. Y.; (2004): *Issue 147: Serial und Parallel For-Each*; WS BPEL Issue List; Zugriff 20.08.04; `http://www.choreology.com/external/WS_BPEL_issues_list.html#Issue147`

[18] Goland, Y. Y.; (2004): *Subject: Issue - 99 - Some proposed wording*; WS BPEL Message List; 5. Mai 2004; Zugriff 14.09.04; `http://lists.oasis-open.org/archives/wsbpel/200405/msg00022.html`

[19] Hecht, M.S.; Ullman, J.; (1972): *Flow Graph Reducibility* SIAM Journal of Computing; Vol. 1, Nr. 2: 188-202

[20] *IBM Terminology - Web Service*; Zugriff 16.10.04, Stand: Juli 2004; `http://www-306.ibm.com/ibm/terminology/`

[21] IBM; Adaptive; Borland; Data Access Technologies; EDS; MEGA; 88 Solutions; (2004): *Buisness Process Defintion Metamodel - Revised Submission to BEI RFP bei/2003-01-06*

[22] Janssen, J.; Corporall, H.; (1997): *Making Graphs Reducible with Controlled Node Splitting*; ACM Transactions on Programming Languages; Vol. 19, Nr. 6: 1031-1052

[23] Johnson, R.; Pearson, D.; Pingali, K.; (1994): *The program structure tree: computing control regions in linear time*; Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation; Orlando, Florida

[24] Jornitz, G. ; Kloppmann, M.; Pfau, G.; Rüttinger, S.; (2003): *WebSphere Application Server Enterprise Process Choreographer Programming Model*; Zugriff 28.10.04; `http://www-106.ibm.com/developerworks/websphere/library/techarticles/wasid/ProcessChoreographerProgrammingModel.html/`

[25] Kas'janov, V.N.; (1975): *Distinguishing Hammocks in a Directed Graph*; Soviet Math. Doklady; Vol. 16, Nr. 5: 448-450

[26] Koehler, J.; Tirenni, G.; Kumaran, S.; (2002): *From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods*; EDOC 2002, 96-106.

[27] Koehler, J.; Hauser, R.; Sendall, S.; Wahler, M.; (2005): *Declarative Techniques for Model-Driven Business Process Integration*; IBM System Journal 44(1)

[28] Lengauer, T.; Tarjan, R.E.; (1979): *A fast algorithm for finding dominators in a flowgraph*; ACM Transactions on Programming Languages and Systems; Vol. 1, Nr. 1: 121-141

[29] Leymann, F.; Roller, D.; Thatte, S.: *Goals of the BPEL4WS Specification*, `http://xml.coverpages.org/BPEL4WS-DesignGoals.pdf`

[30] Leymann, F.; (2001): *Web Services Flow Language (WSL 1.0)*; `http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf/`

[31] Lind, K.; van Norman, E.; (2003): *WebSphere Application Server Enterprise Process Choreographer Programming model for staff resolution*; Juli 2003; Zugriff: 28.10.04; `http://www-106.ibm.com/developerworks/websphere/library/techarticles/wasid/WPC_StaffModel/WPC_StaffModel.html`

[32] Mantell, K.; (2003): *From UML to BPEL, Model Driven Architecture in a Web Service World*; Zugriff 20.07.04, Stand 9.09.2003; http://www-106.ibm.com/developerworks/webservices/library/ws-uml2bpel/

[33] Miksch, S.; (2004): *Unterlagen zur Vorlesung Computerunterstützte (wissensbasierte) Therapieplanung, Part 2 Workflow*; Zugriff 16.09.04; http://www.ifs.tuwien.ac.at/ silvia/wien/therapie/PDF/Part-2.pdf

[34] Object Management Group; (2003): *Unified Modeling Language: Superstructure Version 2.0*

[35] Object Management Group; (2003): *Unified Modeling Language: Infrastructure Version 2.0*

[36] Object Management Group; (2003): *Meta Object Facility (MOF) 2.0 Core Specification*; OMG Document ptc/03-10-04

[37] Object Management Group; (2003): *Buisness Process Defintion Metamodel - Request for Proposal*; OMG Document bei/2003-16-01

[38] Stahlknecht, P.; Hasenkamp, U.; (2002): *Einführung in die Wirtschaftsinformatik*; Berlin; Springer;

[39] Thatte, S.; *XLANG - Web Services for Business Process Design*; Zugriff 17.01.95; http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[40] Trickovic, I.; (2004): *Issue 99: Triggering activities for abstract processes*; WS BPEL Issue List; Zugriff 14.09.04; http://www.choreology.com/external/WS_BPEL_issues_list.html#Issue99

[41] *WebSphere Application Server, Version 5.1.x - The native JMS provider - Writing the WSDL extension*; Zugriff 29.10.04; http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp

[42] Wenzler, A.; (2004): *Web Services und Service Oriented Architecture*; Seminararbeit; Technische Universität Kaiserslautern; Zugriff 20.10.04, Version vom 02.06.04; http://wwwdvs.informatik.uni-kl.de/courses/seminar/SS2004/awenzlera.pdf/

[43] White, S.A.; (2004): *Process Modeling Notations and Workflow Patterns*; Zugriff 19.08.04; http://www.bptrends.com/

[44] Wikipedia: Erläuterung zum Begriff Geschäftsprozessmodellierung; Zugriff 30.06.04; http://de.wikipedia.org/

[45] Wikipedia: Erläuterung zum Begriff Geschäftsprozess; Zugriff 30.06.04; http://de.wikipedia.org/

[46] Williams, M.H.; Ossher, H.L.; (1976): *Conversion of unstructured flow diagrams to structured form*; Department of Computer Science, Rhodes University; Computer Journal Vol. 21 Nr. 2: 161-167

[47] Wohed, P.; van der Aalst, W.M.P.; Dumas, M.; (2004): *Pattern Based Analysis of BPEL4WS*; Department of Computer Science and Systems, Stockholm Universität, Schweden; Department of Technology Management, Universität Eindhofen; Centre of Information Technology Innovation, Queensland Universität

[48] Workflow Management Coalition; (2002); *Workflow Process Definition Interface – XML Process Definition Language*; http://www.wfmc.org/standards/docs/

[49] XML Protocol Working Group; (2003): *Simple Object Access Protocol (SOAP) 1.2 Part 0: Primer, W3C Recommendation 24 June 2003*; Zugriff 25.10.04; `http://www.w3.org/TR/soap/`

[50] Zhang, F.; D'Hollander, E.H.; (2004): *Using Hammock Graphs to Structure Programs*; IEEE Transactions on Software Engineering; Vol. 30, Nr. 4: 231-245

Erklärung

Hiermit erkläre ich, die hier vorliegende Diplomarbeit „Aspekte der Abbildung von Geschäftsprozessen, spezifiziert im Business Process Definition Metamodel, in BPEL4WS“ selbstständig und ohne fremde Hilfe verfasst und nur die angegebenen Hilfsmittel verwendet zu haben.

Leipzig, den 1. Februar 2005

Kirsten Stöhr