Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3324

Eine BPMN-nach-BPEL-Transformation

Omana Omar

Studiengang:	Informatik
Prüfer:	Prof. Dr. Leymann
Betreuer:	Dipl.-Inf. Oliver Kopp
begonnen am:	25. April 2012
beendet am:	10. Oktober 2012
CR-Klassifikation:	H.4.1, K.1

Kurzfassung

Geschäftsprozesse werden typischerweise in BPMN modelliert und mittels BPEL zur Ausführung gebracht. In dieser Arbeit werden bestehende Arbeiten dazu verwendet, um ausführbaree BPMN-Prozessmodelle in ausführbare BPEL-Prozessmodelle zu transformieren. Dabei werden insbesondere der Datenfluss und die Service-Aufrufe berücksichtigt. Die Transformation bildet nicht 1:1 ab, sondern, z. B. im Falle von Schleifen, werden mehrere Konstrukte auf ein Konstrukt abgebildet. Um ein Monitoring auf BPMN-Basis zu ermöglichen, müssen die Zustandsinformationen der laufenden BPEL-Aktivitäten auf das BPMN-Modell abgebildet werden. Die generierung von solchen Abbildungen wird gezeigt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

1 Einleitung

Prozessmodelliersprachen sind ein sehr wichtiges IT Werkzeug in dem Bussiness-Bereich um Interaktionen zwischen den Komponenten oder Elementen darzustellen. Ein BPD (Business Process Diagram) ist die graphische Darstellung eines Business Prozesses. Es ist von Business Process Modelling Notation (BPMN) Kernelemente gebildet, die die Flusskontrolle festlegen. Graphische Sprachen bieten, insbesondere für Nichtfachleute im Bereich der IT, Einfachheit in der Darstellung dieser Interaktionen an. Die Geschäftsprozesse müssen zuerst syntktisch korrekt sein, und andere Korrektheitsregeln entsprechen, um erfolgreich zur Ausführung gebracht zu werden. Beispielsweise muss der Datenfluss im Prozess vollständig definiert sein, der Prozess darf keine Deadlocks haben, die Parameter der Operationen müssen den richtigen Typ haben, usw. Block-orientierte Sprachen bieten für Anwender, die mit Programmiersprachen vertraut sind, eine klare Semantik. Zusätzlich da die Darstellung schon ähnlich zu echten Programmen ist, ist die Transformation zu ausführbaren Code einfacher zu erreichen. [Fou09]

BPMN ist der de-facto Standard, um Geschäftsprozesse graphisch und Graph-orientiert zu modellieren. Damit können alle beteiligten Teilnehmer auf einem Geschäftsprozess mitarbeiten. BPEL bietet einen Standard, um Geschäftsprozesse mit dem Ziel, diese Prozesse später auszuführen, zu modellieren. Zwar bietet BPEL sowohl Graph- als auch blockorientierte Konstruktue an [KMWL09], umfasst der graphische Teil ausschließlich azyklische Graphen. Obwohl im letzten Standard BPMN 2.0 auch die Unterstützung hinzugefügt wurde, Prozesse auszuführen, ist BPEL noch die Sprache, die von vielen Unternehmen zur Ausführung von Geschäftsprozessen benutzt wird [Ley11]. Das Problem damit ist, dass nicht alle Teilnehmer eines Geschäfts alle die technischen Kenntnisse und Grundlagen im Implementierungsbereich haben. Aus diesem Grund, möchten wir einen Transformator, der aus einem BPMN ein entsprechendes BPEL erzeugt, um es danach auszuführen. Dies führt die Vorteile der beiden Sprachen zusammen, bietet Interoperabilität und vereinfacht die Arbeit.

Die Transformation bildet nicht 1:1 ab, sondern, z. B. im Falle von Schleifen, werden mehrere Konstrukte auf ein Konstrukt abgebildet. Um ein Monitoring auf BPMN-Basis zu ermöglichen, müssen die Zustandsinformationen der laufenden BPEL-Aktivitäten auf das BPMN-Modell abgebildet werden. Die generierung von solchen Abbildungen wird gezeigt.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – BPMN, BPEL und Ansätze zur Transformation: Hier werden die Ansätze erläutert, die in dieser Arbeit benutzt werden.

Kapitel 3 – Von BPMN 2.0 nach WS-BPEL 2.0: Hier wird die Transformation von BPMN2.0 nach BPEL2.0 beschreibt

Kapitel 4 – Mapping Sets und State Propagation Rules: erklärt das Verfahren von den State Propagation Sets, das die Elemente mit denen entsprechenden Zustände eines BPMNs in BPEL zurücküberführt.

Kapitel 5 –Architektur und Implementierung beschreibt sowohl die Struktur von den verwendeten und implementierten Klassen als auch die genutzte Libraries.

Kapitel 6 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen

2 BPMN, BPEL und Ansätze zur Transformation

Dieses Kapitel stellt BPMN 2.0, BPEL 2.0 und existierende Ansätze zur Transformation von BPMN nach BPEL vor.

2.1 BPMN 2.0

Die BPMN (Business Process Model and Notation[Obj11]) ist ein Standard der für Geschäftsprozessmodellierung verwendet wird. Er stellt eine standarisierte graphische Notation bereit, um Geschäftsprozese in Geschäftsprozessediagrammen zu definieren und um Workflows zu modellieren, die von einer Workflow-Engine zur Ausführung gebracht werden können. BPMN ist derzeit eine wichtige Sprache sowohl für technisch-orientierte und geschäfts-orientierte Nutzer, da sie nach eigenen Angaben eine intuitive Notation anbietet, um komplizierte Semantik darzustellen.

Es gibt 3 Grundtypen von Modellen die repräsentiert werden können: Private (innere) Geschäftsprozesse, Abstrakte (öffentliche) Prozesse und Kollaboration (globale) Prozesse:

- Private (innere) Geschäftsprozesse Private Geschäftsprozesse sind solche, die in einer bestimmten Organisation verwendet werden und sind allgemein Workflowprozesse oder BPM-Prozesse genannt.
- Abstrakte (öffentliche) Prozesse Diese repräsentieren die Interaktionen zwischen private Geschäftsprozesse und andere Prozesse oder Teilnehmer. In diesen Prozessen werden nur die Aktivitäten inkludiert, die außer den privaten Geschäftsprozessen kommunizieren.
- Kollaboration (globale) Prozesse Ein Kollaboration Prozess beschreibt die Interaktionen zwischen 2 oder mehrere Organisationen. Diese Interaktionen sind die Aktivitäten, die den Nachrichtenaustausch zwischen den beteiligten Teilnehmern repräsentieren.
- Koreografien Eine Koreografie ist eine Definition vom erwarteten Verhalten zwischen interagierende Teilnehmer. Die Koreografie sieht ähnlich aus wie ein Privater Geschäftsprozess, da sie aus Aktivitäten, Ereignisse und Gateways besteht. Allerdings sind sie verschieden, da die zur Koreografie gehörende Elemente bezieht zwei oder mehr Teilnehmer ein. ...

BPMN hat mehr als 100 graphische Elemente die in die folgende 5 Kategorien klassifiziert werden: *Flussobjekte*, *Datenobjekte*, *Connecting objects*, *Swimlanes* und *Artefakte*.

Flussobjekte definieren das Verhalten eines BPD Prozesses. Es gibt 3 Typen von *Flussobjekte*: Ereignisse, Gateways und Aktivitäten. Ein Ereignis wird durch einen Kreis dargestellt und ist ein Ding, das während der Ausführung eines Prozesses, passiert. Sie passieren wegen einer Ursache (*trigger*) oder generieren eine Folge (Ergebnis) die den Fluss eines Prozesses auswirkt. Ereignisse können sich am Anfang (Startereignisse), in der Mitte (Zwischenereignisse) oder am Ende (Endereignisse) des Prozesses befinden. *Gateways* werden durch eine Raute repräsentiert und werden benutzt um Entscheidungen zu bestimmen, Pfade zu gabeln und zusammenzuführen abhängig von dem Gatewaytyp (Exklusives, Paralleles, Inklusives und Komplexes). *Aktivitäten* werden durch ein Viereck dargestellt und die repräsentieren die Arbeit die in einem Prozess ausgeführt wird. Eine Aktivität kann atomare (*Task*) oder zusammengesetzt (SubProzess) sein. *Artefakte* zeigen zusätzliche Information im Diagramm, die nicht direkt mit dem Sequenzfluss oder Nachrichtenfluss des Prozesses zusammenhängen. BPMN stellt *Datenobjekte*, *Gruppierungen* und *freie Anmerkungen* bereit. *Datenobjekte* zeigen, welche Daten erforderlich für die Aktivitäten sind und welche von Aktivitäten ausgegeben werden. Die Richtung der Assoziationen weißt hin, ob die Daten als Eingabe oder Ausgabe verwendet werden. Eine *Gruppierung* wird durch ein abgerundetes Rechteck mit einer gestrichelten Linie dargestellt. Die gruppieren Aktivitäten zur Dokumentation oder Analysezwecke. *Freie Anmerkungen* bieten zusätzliche Text-Informationen für den Leser vom BPD.

Flussobjekte und *Artefakte* werden mit Hilfe von Verbindungsobjekte verbunden. Es gibt verschiedene Arten von Verbindern, die Flussobjekte und Artefakten verbinden. Ein Sequenzfluss wird durch eine durchgezogene Linie mit einer durchgehenden Pfeilspitze dargestellt. Er definiert die Abfolge der Ausführung der Aktivitäten eines Prozesses. Ein *Nachrichtenfluss* wird durch eine gestrichelte Linie mit einer offenen Pfeilspitze dargestellt. Er symbolisiert den Informationsaustausch und die Kommunikation zwischen getrennten Teilnehmern, d. h. dass er die Entsendung und der Empfang einer Nachricht, durch die Verbindung der Nachrichtenfluss mit der sendenden und der empfangenden Aktivität, zeigt. Eine Assoziation wird durch eine gepunktete Linie dargestellt und verknüpft Artefakte mit Flow-Objekte.

2.2 WS-BPEL 2.0

BPEL oder auch WS-BPEL (Business Process Execution Language für Webservices), ist eine domänenspezifische, imperative XML-basierte Programmiersprache zur Definition und Beschreibung von ausführbaren Geschäftsprozessen, die den Nachrichtenaustausch mit anderen Systemen beteiligen. Obwohl normalerweise die typische BPEL-Anwendung die Modellierung von ausführbaren Prozessen ist, kann BPEL auch benutzt werden, um nicht-ausführende Prozesse zu erstellen. Diese „abstrakten“ Prozesse können für verschiedene Anwendungen verwendet sein. Den Zweck der Anwendung definiert das „abstract process profile“.

Listing 2.1 BPMN2.0 Beispiel das Web-Service Tasks enthält

```
<ns8:serviceTask xmlns:ns8="http://www.omg.org/spec/BPMN/20100524/MODEL"
    xmlns=""
    completionQuantity="1" id="sid-8C884956-14EA-4EA2-B530-C1135BF0AE61"
    implementation="webService" isForCompensation="false"
    name="Service Call Amazon" operationRef="ProduktBestellung"
    startQuantity="1">
</ns8:serviceTask>
```

Eine Prozessdefinition besteht aus einem BPEL-Dokument und einem oder mehreren WSDL-Dokumenten zusammen. Die WSDL-Dokumente definieren die Schnittstellen der zu orchestrierenden Services wie auch die Schnittstelle, unter der der Prozess selbst erreichbar ist. Für die Verwendung in BPEL sind anfänglich nur die abstrakten Teile der WSDL-Dokumente relevant, die sind die Definitionen von Nachrichtentypen und die Gruppierung von Operationen zu Porttypen.

Das Wurzelelement eines WS-BPEL-Prozesses ist das <process>-Element im BPEL-2.0-Namensraum, das über das Atributpaar *name* und *targetNamespace* ein BPEL-Prozessmodell identifiziert. Innerhalb des <process>-Elements befindet sich das <import>-Element, das es erlaubt externe Artefakte, typischerweise WSDL- und XSD-Dateien zu importieren. Als nächstes werden die *Partnerlinks* und Variablen definiert. Die *Partnerlinks* sind Verträge die verbinden, die Schnittstelle die der Prozess von dem Partner verlangt, mit der Schnittstelle, die der Partner im Gegenzug vom Prozess erwartet. Die stellen eine Punkt-zu-Punkt-Verbindung zwischen den Porttypen beider Services dar.

2.3 Ansätze zur Transformation

Prozessmodelle dokumentieren nicht nur Prozesse in Unternehmen, sondern sollen auch selbst zur Ausführung gebracht werden. Dies kann einerseits durch eine Workflow-Engine umgesetzt werden: Die Workflow-Engine interpretiert das Prozessmodell und führt es aus. Andererseits kann es durch eine Transformation geschehen, die einem Prozessmodell in eine Sprache übersetzt, die von einer Workflow-Engine verstanden wird.

Für die Zwecke dieser Arbeit, wird eine Lösung vorgeschlagen, die ein Prozessmodell von BPMN Sprache in BPEL transformiert wird, das von einer Workflow-Engine verstanden ist, um es später auszuführen. Im Folgenden werden die Transformationsansätze von verschiedenen Autoren beschrieben:

Vanhatalo et. al [VVK08] beschreiben die erste Schritte vom Transformation-Ansatz, der einen Prozess-Modell in einen generalisierten Workflow-Graphen übersetzt, wobei ein generalisierter Worfklow-Graph ein Graph ist, der mehrere Start- und Endereignisse haben kann. Dafür verwenden sie eine erste Übersetzung, die jedes BPMN-Element zu einem oder mehreren Workflow-Graph-Elemente zuordnet. Hier ist jede Kante i vom BPMN Diagramm D in eine

Kante i' im entsprechenden Workflow-Graph G. Ein paralleles Gateway p von D, wird in ein paralleles Gateway p' zugeordnet. Diese parallele Gateways haben auch entsprechende eingehende und ausgehende Kanten Aktivität, die direkt vom D in G zugeordnet werden. Abbildung 2.1 zeigt diese Zuordnung.

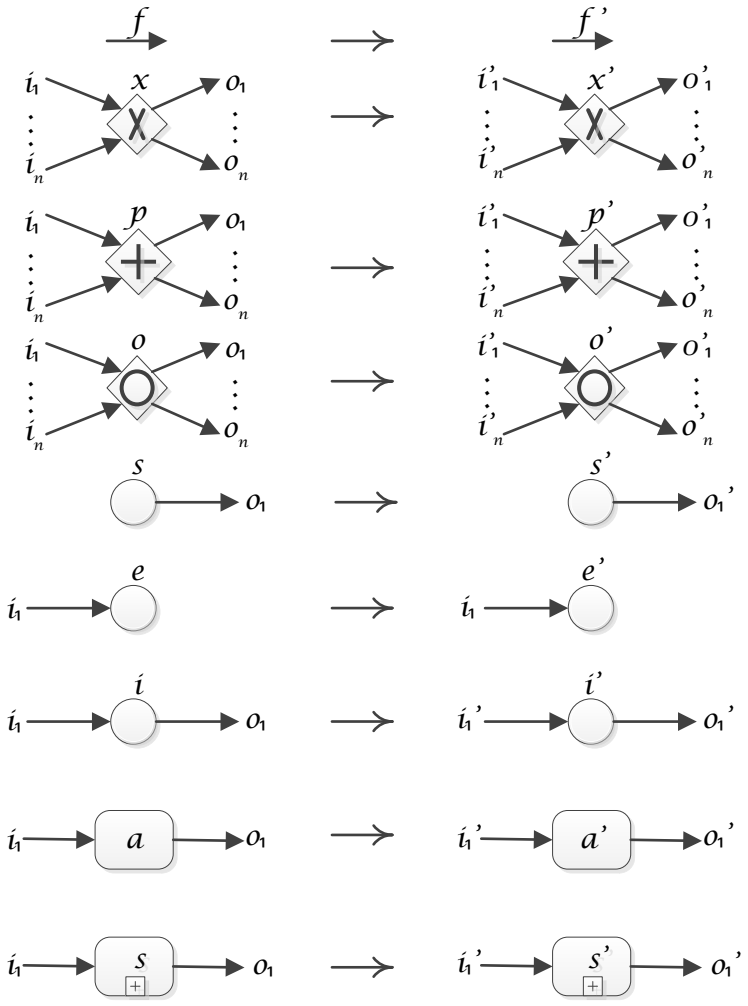


Abbildung 2.1: 1:1 Zuordnung von BPMN-Elemente nach Workflow-Elemente

Analog gilt dieses Zuordnungsverfahren für exklusive Gateways, inklusive Gateways und Ereignisse. Wobei es angenommen wird, dass jedes Ereignis höchstens eine eingehende Kante und eine ausgehende Kante hat. Dasselbe Zuordnungsverfahren wird für eine Aktivität/-Teilprozess verwendet, wenn diese höchstens eine eingehende Kante und eine ausgehende

Kante hat. Anderenfalls, wenn eine Aktivität/Teilprozess mehrere eingehende Kanten oder ausgehende Kanten hat, folgt die Zuordnung so:

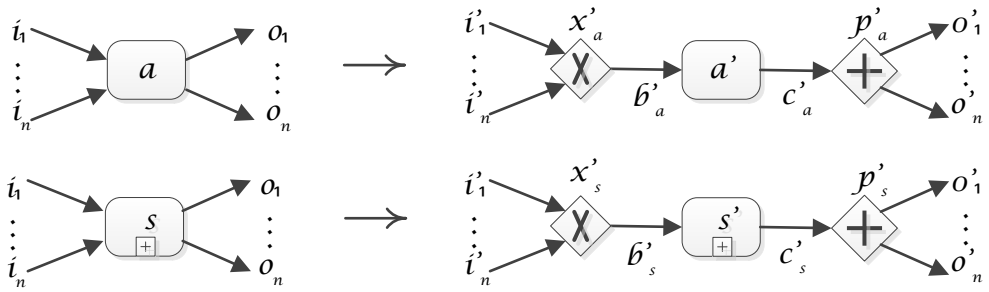


Abbildung 2.2: BPMN-Elemente die eine 1:n Zuordnung nach Workflow-Elemente

Die Aktivität a von D, wird in eine Aktivität a' in G zugeordnet. Zusätzlich wenn die Aktivität mehrere eingehende Kanten hat, wird ein exklusives Gateway x'_a durch eine neue Kante vor a' in G hinzugefügt, und so werden die eingehende Kanten $(i_1 \dots i_n)$ von a, die eingehende Kanten von x'_a $(i'_1 ... i'_n)$ entsprechen. Ähnlicherweise, wenn eine Aktivität a mehrere ausgehende Kanten hat, wird ein zusätzliches paralleles Gateway p'_a durch eine neue Kante nach a' in G hinzugefügt und jede ausgehende Kante $(o_1 \dots o_n)$ von s wird eine ausgehende Kante von p'_a $(o'_1 \dots o'_n)$ entsprechen. Der letzte Ansatz gilt für Teilprozesse auch.

Sobald der erste Schritt durchgeführt wird, und man den dem BPMN Diagramm entsprechenden Workflow-Graph erhält, sollte dieser Workflow-Graph durch Compiler Theorie Techniken unterteilt werden. Diese Techniken werden wiederum auf Dekompositionstechniken von ungerichteten Graphen der Graph Theorie beruht. Die Unterteilung soll durchgeführt werden, um davon strukturelle Informationen zu bekommen und die Flusskontrolle des Workflows zu analysieren.

Nach Vanhatalo et al. [VVK08] gibt es zwei Arten von Unterteilungen, die zwei Typen von Parse-Bäumen erzeugen: *Normal Process Structure Trees* (NPST) und *Refined Process Structure Trees* (RPST).

Der NPST ist ein hierarchischer Baum, der aus *N-Fragments* von dem G Graph besteht, während der RPST ein hierarchischer Baum ist, der aus *R-Fragments* besteht. Wo ein *R-Fragment* eine Menge von Kanten ist, die genau zwei Grenzeknoten hat (ein Eingangs- und ein Ausgangsknoten) und die einen verbundenen Teilgraphen bilden. Ein *N-Fragment* anderseits ist eine Menge von Knoten die auch verbunden sind.

Der *Refined Process Structure Tree* hat Vorteile um bestimmte Aufteile zu bieten, die in viele Anwendungen wie BPMN-BPEL Transformatoren helfen. Er wird benutzt um einen *Process Tree* darzustellen, weil dieser eine strukturierter Darstellung und mehr Information als andere Parse-Bäume präsentiert. Er verschafft eine feingranulare Unterteilung und hilft, eine direkte Transformation von BPMN in BPEL zu geben. Die Wurzel von einem RPST ist der größte Ausschnitt, der alle die *R-Fragments* eines bestimmten Graphs G enthält. Die zum RPST gehörende *R-Fragments* müssen „objective“ sein. „Objective“ bedeutet, dass jedes *R-Fragment* nicht mit einem anderen überlappt.

Ouyang et al. [ODHA06] stellen einen Ansatz vor, um ein BPD in ein BPEL-Prozess zu überführen. Zu diesem Zweck, wird ein BPD in Komponenten, die BPD Untermengen darstellen, unterteilt. Diese Komponenten sind durch einen Einzeleingangsknoten und einen Einzelausgangsknoten gekennzeichnet. Wichtig zu beachten ist, dass eine Komponente zumindest aus einem Element besteht, d. h. eine Kante mit den bestimmten Quell- und Senkenknoten. Komponenten definieren auch drei Grundstrukturen, die direkt in BPEL überführt werden können. Diese drei Strukturen sind: *well-structured patterns*, *quasi-structured patterns* und *generalised FLOW-patterns*. Das *well-structured pattern* besteht aus allen Komponenten, die keinen Deadlock oder keine mehrfache Instanzen der Ausführung von einer selben Aktivität verursachen [KHB00] (Kiepueszewski). Diese können in eine von den folgenden BPEL Strukturen direkt zugeordnet werden: sequence (Abbildung 2.3), flow (Abbildung 2.5), if(Abbildung 2.4), pick (Abbildung 2.6) und while (Abbildung 2.7). Diese Zuordnung kann gemacht werden, wenn beide Strukturen Verhaltensweise entsprechen.

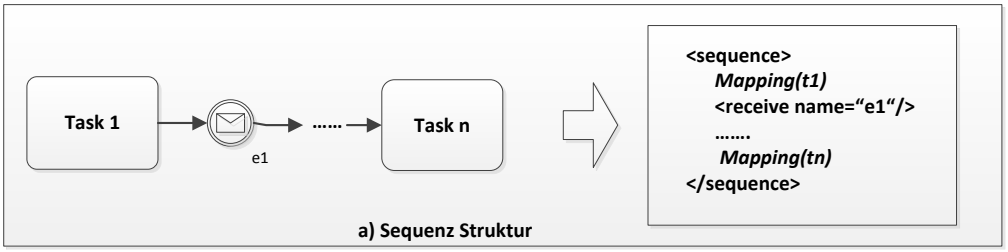


Abbildung 2.3: Well Structured Pattern entsprechend der Sequenz

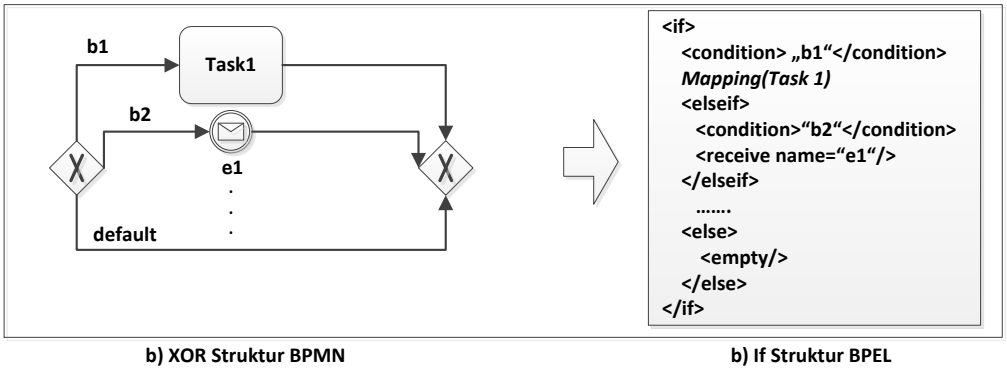


Abbildung 2.4: Well Structured Pattern entsprechend dem Switch

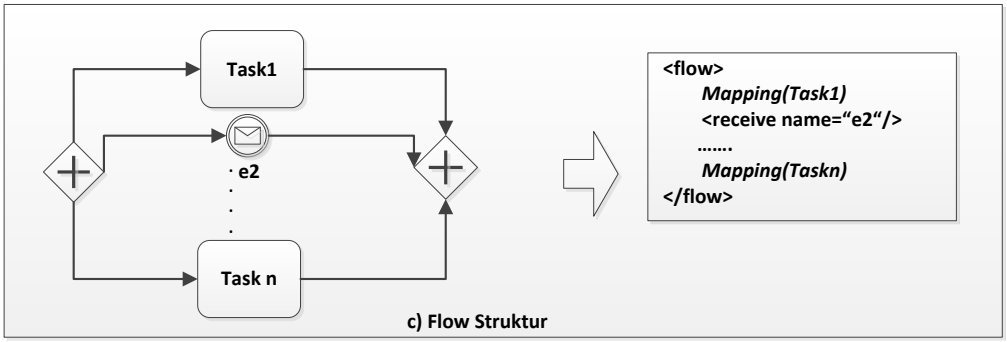


Abbildung 2.5: Well Structured Pattern entsprechend dem Flow

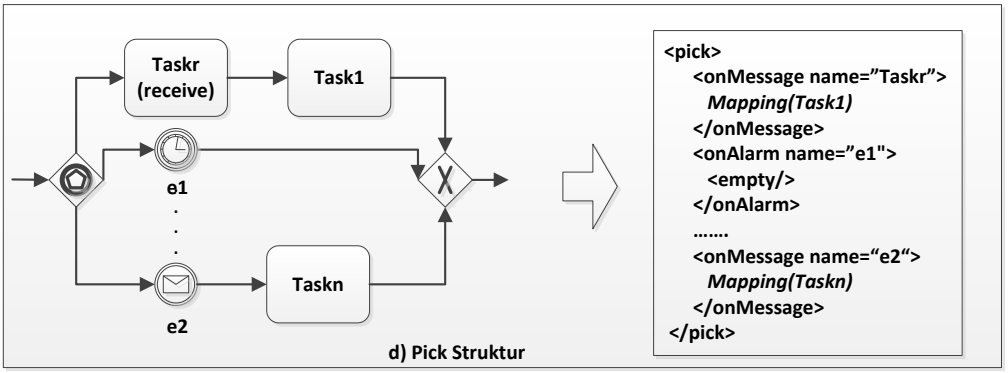


Abbildung 2.6: Well Structured Pattern entsprechend der Pick

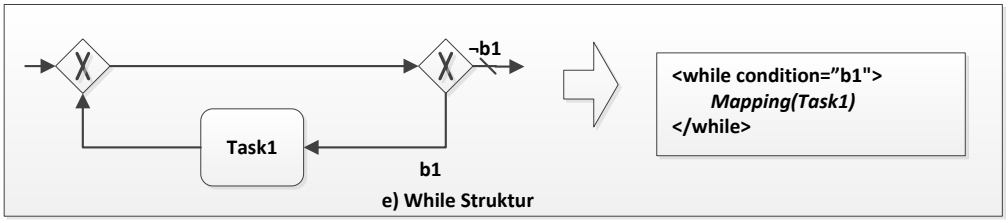


Abbildung 2.7: Well Structured Pattern entsprechend der While

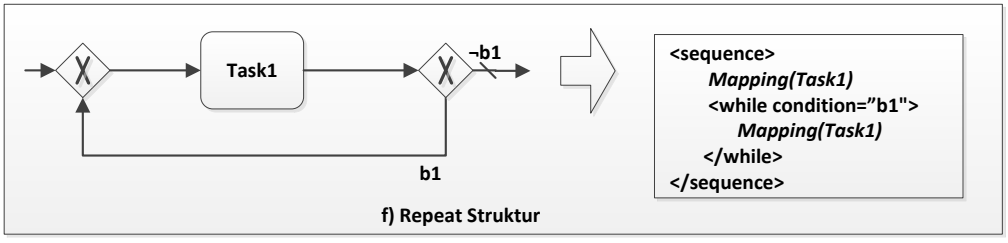


Abbildung 2.8: Well Structured Pattern entsprechend der Repeat

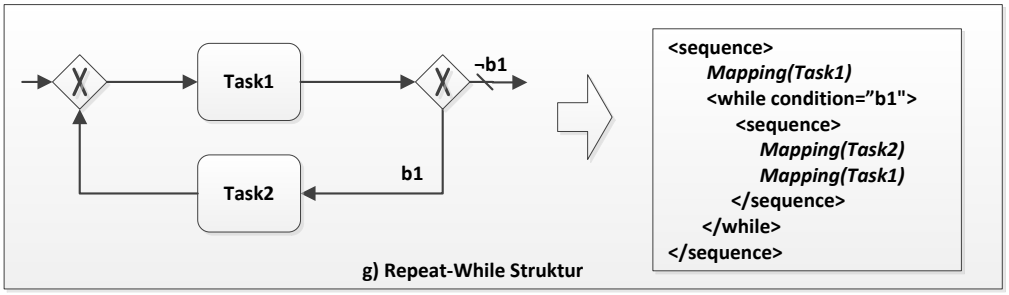
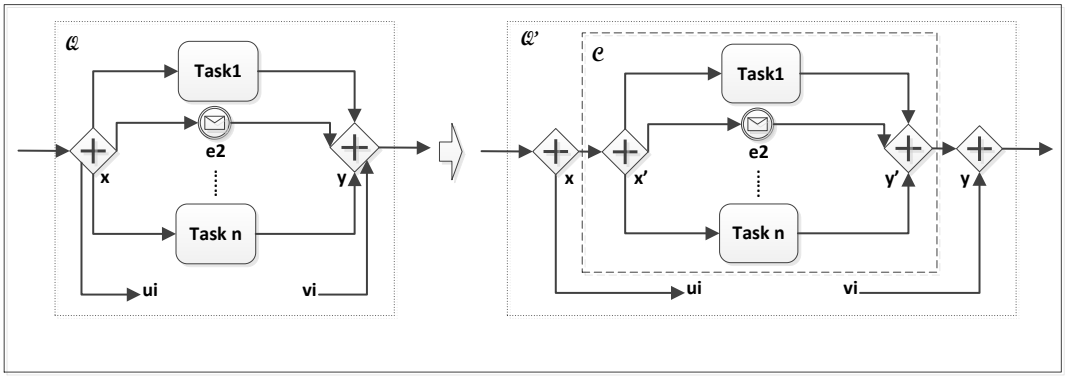


Abbildung 2.9: Well Structured Pattern entsprechend der Repeat-while

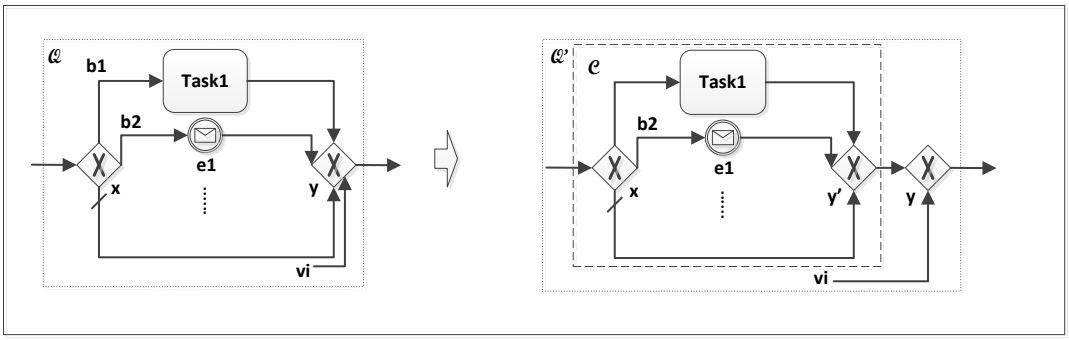
Die *quasi-structured components* sind alle die Komponenten, die aus nicht zusammengehörige Paare von Split- und Join-Gateways bestehen. Diese Teilgraphen müssen zuerst in *Well-Structured Patterns* transformiert werden, um ihre Verständnis und Analyse zu erleichtern. Diese erlauben eine direkter Transformation in BPEL zu leisten.

Ouyang et al. unterteilen die *quasi-structured components* in 3 Typen: FLOW, SWITCH und PICK, abhängig von der Art des Gateways, das die Komponente als Eingangsknoten hat (Paralleles, Exklusives beziehungsweise Ereignis-basiertes). Diese 3 Fälle die Ouyang vorschlägt, sind in dieser Arbeit durch die Analyse von der Typ des Eingangs- und des Ausgangs-Gateways berücksichtigt, wenn die Transformation von *quasi-structured components* nach *Well-Structured Patterns* durchgeführt wird.

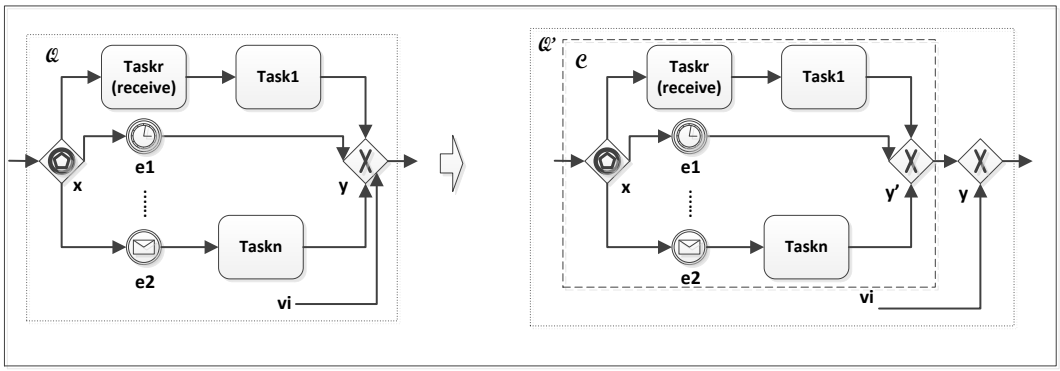
Vanhatalo, bietet einen Ansatz vor, in dem eine Refaktorisierung von diesen Strukturen durchgeführt wird, um einen besser strukturierten Graph G* zu bekommen. Diese Refaktorisierung besteht in der Identifizierung von Gateways, die Endeknoten von zwei verschiedenen Strukturen sind, Startknoten von zwei verschiedenen Strukturen sind oder die Endeknoten einer Struktur und Startknoten einer anderen sind. Danach sollen diese in zwei verschiedene Gateways gespaltet werden (durch die Hinzufügung eines neues Gateways), um die Paare von Gateways auszugleichen. Dieses Verfahren wird von Vanhatalo et al. [VVK08] "gültige Erweiterung".



(a) Quasi FLOW Pattern → Well Structured FLOW



(b) Quasi SWITCH/IF Pattern → Well Structured SWITCH/IF



(c) Quasi PICK Pattern → Well Structured PICK

Abbildung 2.10: Teilweise strukturierte Strukturen

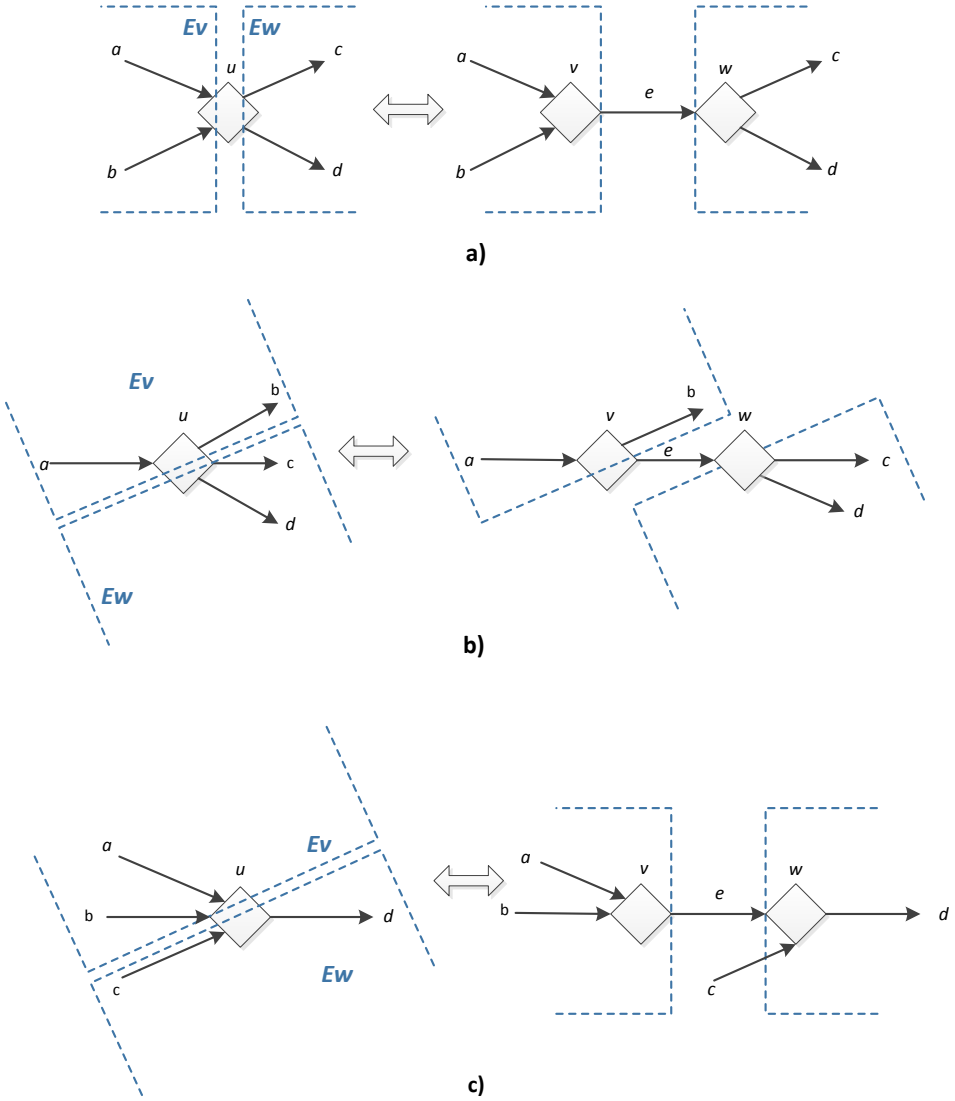


Abbildung 2.11: a), b) und c) gültige Erweiterungen

Die letzte Struktur ist die *Generalised Flow Pattern*, die keine bestimmte Strukturierung hat und die in keiner von den oben genannten Kategorie von Strukturen fällt. Diese Strukturen sind azyklisch und haben nur Parallele Gateways und keine andere enthaltene Komponente.

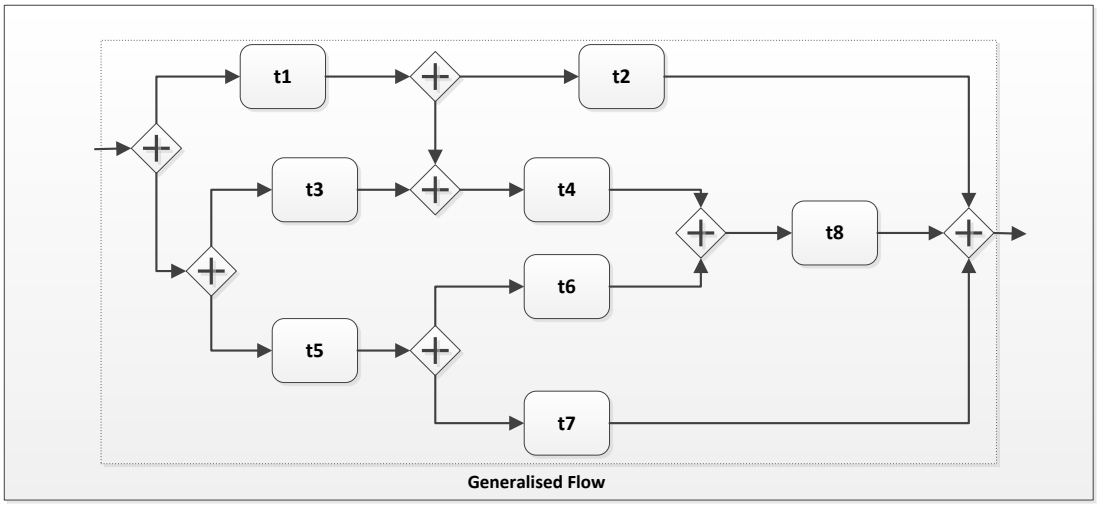


Abbildung 2.12: Example of Generalised Flow

Die Idee dieses Ansatz von Ouyang ist, in dem *Process tree* die Kanten, die ein Fork Gateway x und ein Join Gateway y direkt verknüpfen, zu identifizieren, um die zu löschen. Danach werden alle die Gateways gelöscht, die keine Verkehrslenkung-Funktion durführen, d. h. die Eingangs- und Ausgangsgrad von 1 haben. Einschließend wird einen Link von dem Vorgänger des gelöschten Gateways nach dem Nachfolger des gelöschten Gateways hinzugefügt. Mendling et al. [MR06] anderseits vorschlagen, die verbleibende Gateways durch eine Empty Aktivität und jede Kontrollfluss-Kante durch einen entsprechenden Link zu ersetzen.

Sobald die Komponenten identifiziert und transformiert sind, wird jede Komponente mittels der *Fold* Funktion durch seine entsprechende Übersetzung (ein BPEL Konstrukt) ersetzt. Diese Funktion wird rekursiv (bottom-up) ausgeführt.

Die wichtige *R-Fragments* für die Analyse und Aufteilung des RPSTs, um die nachfolgende Transformation zu generieren, sind:

- Sequenz oder Polygon: Repräsentiert eine Reihenfolge von R-Fragments, diese soll maximal sein, d. h. dass die, nicht in einer anderen Sequenz enthalten wird, um das „Objektivitätsprinzip“ zu bewahren.
- Bond: Ein Bond von u nach v, Eingangsknoten bzw. Ausgangsknoten, ist die Vereinigung von zumindest 2 Branches. Wo ein Branch, eine Sequenz von *R-Fragments* von u nach v oder von v nach u ist.
- Trivial: Ein Trivial besteht aus einer Kante die 2 Knoten im *Process Tree* verbindet

- Rigid: Diese haben keine bestimmte Strukturierung und verbinden normalerweise Gateways, die keine Umlenkungsfunktion übernehmen, oder Gateways, die beide Funktionalitäten von Split und Join haben. Diese Strukturen können homogen, d. h. nur XOR Gateways oder AND Gateways haben, oder heterogen in denen eine Mischung zwischen XOR und AND Gateways sich findet [DGBP10]. Für diese Arbeit werden nur die homogene AND Generalisierte Flows berücksichtigt.

Garcia-Bañuelos et al. [Garo8] leistet einen Beitrag zu dieser Arbeit, mit der Beschreibung einer Methode, die die Identifizierung der BPMN-Mustern systematisiert, durch die Unterteilung des Workflows in *Single Entry Single Exit* SESE Regionen und Sammlung von Erreichtbarkeitsinformationen durch das Lösen von Datenfluss-Gleichungen. Im Gegensatz zu anderen Arbeiten, in denen die Teilgraphen durch die Analyse des Graphen identifiziert sind.

Er bringt vor, dass der PST *Process Structured Tree* durch Preorder Traversierung durchgelaufen werden soll. Dabei wird jede triviale Komponente ohne weitere besondere Verarbeitungen geskippt und die anderen Komponenten im BPMN Diagramm durch entsprechende BPEL-Konstrukte ersetzt werden. Um diese Strukturen zu identifizieren, soll der Graph in Komponenten aufgeteilt werden. Danach wird eine Menge von Datenflussgleichungen benutzt, um Information zu bekommen, und später die Komponenten zu identifizieren. Er präsentiert verschiedene wichtige Strukturen:

- Strukturierte Zyklen: Sie sind die Muster, die ein XOR Gateway als Eingangsknoten, ein XOR als Ausgangsknoten und einen Weg vom Ausgangsknoten nach dem Eingangsknoten haben.
- Strukturierte Blöcke: Enthält alle die Komponenten, die ein Split-Gateway als Eingangsknoten und als Ausgangsknoten haben. Diese können in die folgende BPEL-Strukturen direkt abgebildet werden: *FLOW* wenn beide Gateways (Eingang und Ausgang) AND sind, *IF* wenn beide Gateways XOR sind und *PICK* wenn das Eingangsgateway ein Ereignisbasierte XOR ist und das Ausgangsgateway ein XOR ist.
- Unstrukturierte Blöcke: Sind alle azyklischen Blöcke mit beliebiger Anordnung. Sie haben als Eingangsknoten und Ausgangsknoten Parallele Gateways, aber dazwischen mehr Gateways, die mit einender und mit anderen Teilen der Struktur verbunden sind. Diese werden von Ouyang als „Generalisierte Flows“ bezeichnet.
- Sequenzen: Ist die Zusammenfügung von Komponenten, die sich in einer Reihenfolge finden.
- Beliebige Zyklen: Diese sind Zyklen, die mehrere Eingangsknoten und Ausgangsknoten haben.

Stein et al. [SKI09] klassifizieren die Modellierungssprachen in zwei Typen, abhängig von der Lesbarkeit des Models und von der Art der Darstellung, die benutzt wird. BPMLs *Business Process Modelling Languages* wie BPMN oder EPC, die hauptsächlich graphisch strukturiert sind, in denen z. B. der Kontrolfluss wird durch Kanten bestimmt, die die Ausführungslogik zwischen den Knoten repräsentieren. Anderseits definieren die Blockorientierte Sprachen

den Kontrolfluss durch eingebettete Elemente, die Sequenz, Konkurrenz, Verzweigungen und Zyklen repräsentieren. Laut Stein kann die Lücke zwischen Geschäftsmodelle und IT-Modelle durch Transformationen geschlossen werden. Er gibt an, dass es bezüglich der Granularität der Transformation, zwei Arten von Transformationen gibt. Die horizontale, die Modelle auf derselben Abstraktionsebene transformieren und die vertikale Transformationen, die das Eingangsmodell zu einem detaillierteren Niveau durch die Hinzufügung von zusätzlicher Information verfeinern. Gemäß Stein, gibt es verschiedene Ansätze die benutzt werden können, um eine Transformation von einer BPML in eine Blockorientierte Sprache durchzuführen. Der wichtigste Ansatz für den Zweck diese Arbeit ist der *Control Flow Centred*, in dem Compilertheorie Techniken verwendet werden, um ein Prozess Diagramm in BPEL Code zu transformieren. Dieser Ansatz hat nach Mendling et al. [MLZ06] vier Vorgehensweisen: *Element-Preservation*, ist die einfachste, da mithilfe dieser Methode wird jedes Element des Workflow-Graphs in ein entsprechendes BPEL-Element zugeordnet z. B. *Flow-Elements* sind in Aktivitäten zugeordnet, die in einem Flow-Konstrukt geschachtelt werden, und die Kanten sind in Links zugeordnet um den Fluss zuleiten. Das resultierende BPEL wird ist sehr ähnlich dem ursprünglichen Workflow-Graph, da die Zuordnung 1:1 ist, aber der gehörende Code wird kaum lesbar. Die zweite Vorgehensweise ist *Element-Minimization*, in der die Idee ist, leere Aktivitäten (*Empty Activities*) zu löschen, die aus Gateways erzeugt wurden und stattdessen das Split- und Join-Verhalten durch Links bzw. Join-Conditions modellieren. *Structure-Identification* geht darum, dass im *Workflow-Graph* bestimmte Muster identifiziert werden, um die später in strukturierte BPEL-Aktivitäten zu überführen. Schließlich die *Structure-Maximization* benutzt die *Structure-Identification* Vorgehensweise rekursiv um strukturierte Muster zu identifizieren, die in Unstrukturierte Muster enthalten sind. Diese Methode ist normalerweise zusammen mit der *Element-Preservation* benutzt um die Transformation in BPEL zu leisten. In dieser Arbeit wird die *Structure-Maximization* Vorgehensweise benutzt.

Dieses Kapitel präsentierte die wichtigen Ansätze zur Analyse von Strukturen des BPMN-Prozesses für ihre spätere Verarbeitung. In einem Vordergrund sind die Ansätze, die das BPMN-File in eine grafische Darstellung transformiert, die leichter zu analysieren und in Komponenten aufzuteilen ist. Auf einer anderen Ebene sind die Ansätze, die der graphisch dargestellte Prozess analysiert um die Strukturen, die transformiert werden, zu erhalten. Diese Gruppe von Ansätze beinhaltet eine Kategorisierung von Strukturen in *well-structured patterns*, *quasi-structured patterns* oder *generalised FLOW-patterns*, die abhängig von ihrer Komplexität und der Möglichkeit sind, dass diese Strukturen Deadlocks haben oder gleichzeitige mehrfache Instanzen der Ausführung von Einer selben Aktivität durchführen. Wiederum wird einen Ansatz hier einbezogen, der die Strukturen in einem RPST Baum identifiziert, dieses Identifizierungsverfahren ist in Bezug auf die Elemente-Typen, diese Bäume haben. Dieser Ansatz orientiert sich an der Struktur, die in der Implementierung verwendet ist. Schließlich wird es hingewiesen, welcher Typ von Transformation nach Steins Klassifizierung benutzt wird.

3 Von BPMN 2.0 nach WS-BPEL 2.0

In diesem Kapitel wird die eigentliche Transformation von BPMN 2.0 nach WS-BPEL 2.0 vorgestellt. Dabei wird als Beispiel ein Deployment-Prozess [Wagner12, Breitenbücher12] für die SugarCRM-Applikation verwendet. Die Struktur der Vorstellung dieses Transformationsverfahren wird so aufgeteilt: Die erste Ebene entspricht dem Typ des basischen BPMN Element zu übersetzen (Aktivitäten, Sequenzflüsse, Ereignisse und Gateways), die nachfolgenden Ebenen der Transformation entsprechen den Subtypen, die diese Elemente haben (wie im Falle von Ereignissen oder Aktivitäten) und die Art der Struktur, die diese Elemente nach Steins Klassifikation erzeugen. Im Falle der Gateways: *well-structured*, *quasi-structured* und *Generalized Flows*

Die Transformation geht davon aus, dass ein BPMN-Modell vorliegt, das an sich ausführbar ist. Das bedeutet insbesondere, dass alle Datenabhängigkeiten modelliert sind. Der erste Schritt von der Transformation von einem BPMN2.0 in ein BPEL besteht in dem Parsen des BPMN Files. Alle die Flow Elemente sind in dem „definitions“ Element enthalten. Das Parsen-Verfahren beginnt mit dem Startknoten des BPMNs. Für jedes gefundene Fluss-Element im File, müssen die ausgehenden Sequenzflüsse weiter analysiert werden um die Senkenelemente zu bekommen und so wird jedes Kind durch Tiefensuche besucht, während der Graph erstellt wird. Falls das gefundene Element, ein Teilprozess ist, werden die in diesem Teilprozess enthaltenden Aktivitäten in einem Teilbaum gespeichert.

Die BPMN-elemente werden in einen Baum (BPMNProcessTree) überführt. Dieser Baum besteht hauptsächlich aus Knoten (WFNode) und Kanten (WFEdge). In diesem Baum enthält jeder Knoten als Attribute: das entsprechende BPMN-element, ein Flag das hinweist ob der Knoten zu dem Prozesshauptfluss gehört (andernfalls sollte der Knoten zu einem Handler- oder zu einem angehefteten Ereignisfluss gehören), einen Baum mit der Transformation der enthaltende Elemente, im Falle der Elternknoten ein Teilprozess ist, und als letztes eine Liste von Graphen, in der jeder Graph einen an der Aktivität angehefteten Ereignisfluss repräsentiert.

Um die Überführung von den im Graph identifizierten Strukturen durchzuführen, wird diesen Graph durch die Aufteilung in Komponenten in einen RPST transformiert. Sobald man den RPST hat, wird eine Funktion *BpmnProctree2BpelModelPart* den R-Fragments Baum durch Tiefsuche durchlaufen. Diese Funktion führt die “Folding” oder Transformation von jeder entsprechenden Komponente durch. Stufenweise während die Kinder eines Knotens überführt werden, wird diese Transformation zu dem Transformation des Vaters durch Backtracking hinzugefügt.

3.1 Mapping von einzelnen Aktivitäten

Die Aktivitäten sind der Kern des Prozesses. Diese geben etwas an, das vom Prozess getan werden soll, um die gewünschte Leistung durchführen zu können. Um die Zuordnung der Aktivitäten durchzuführen, werden die Komponenten analysiert auf der Suche nach trivialen Strukturen, d. h. Strukturen die aus einer Kante und 2 Knoten (Startknoten und Endeknoten) bestehen. Der Eingangsknoten wird überprüft, um zu bestätigen, ob das enthaltende Element den Typ „Task“ hat, und welche der Subtyp und Eigenschaften sind (Service Task, Receive Task, senden Aufgabe, No Task-Typ) um die entweder in „invoke“, „receive“ oder „empty“ zu überführen.

Abbildung 3.1: Aktivitäten Typen

3.1.1 Service Aufrufe in BPMN

Ein Service-Aufruf kann durch das <ServiceTask>-Konstrukt in BPMN durchgeführt werden. Diese ServiceTask übernimmt die Funktionalität eine bestimmte Operation aufzurufen. Als Attribute hat die ServiceTask *implementation*, das spezifiziert welche Technologie benutzt wird, um die Nachrichten zu senden und empfangen, der andere *operationRef* referenziert eine Operation in einem services *interface*.

Innerhalb der Servicetask als verschachteltes Element, befindet sich die *ioSpecification* Definition, die Deklarationen von Elementen und Daten enthält, die intern als Eingabe- und Rückgabewerte der Servicesaufgabe nach der Durchführung des Aufrufs verwendet werden. Diese Elemente sind nicht physisch in dem BPMN-Diagramm repräsentiert, da sie innerhalb der Aufgabe der Aktivität benutzt werden. Sie referenzieren durch das *itemSubjectRef* Attribut ein DataObjekt das im selben BPMN File deklariert ist und im WSDL File definiert ist zusätzlich mit der Struktur und die enthaltenen Elemente.

Außerdem gibt's auch das DataInputAssociation-Element, das die Korrespondenz zwischen den äußeren Daten, die der Service-Aufruf als Eingabeparameter hat und den inneren der Aktivität definierten Eingabe-Daten. Diese Referenzierung wird das Quellendatenobjekt mit dem Zieldatenobjekt durch <SourceRef> und <TargetRef> verbinden. In diesen Elementen werden die „ID“ von den Datenobjekten erhalten.

In diesem Element wird es genau durch das <assignment>-Element präzisiert, wie die Zuweisung gemacht werden muss, d. h. wie aus den Eingabewerten des ServiceTask, die innen von der Aktivität des ServiceTask benötigten Daten erzeugt werden. Dies ist hilfreich wenn das Eingabe-Daten-Objekt vom ServiceTask mehrere enthaltene Elemente hat, um hinzudeuten welche Elemente miteinander entsprechen. Analog zu DataInputAssociation gibt es das DataOutputAssociation-Element, das die Korrespondenz hindeutet, zwischen den äußeren Daten, die der Service-Aufruf als Ausgabeparameter hat und den inneren der Aktivität definierten Ausgabe-Daten.

Eine zu einer ServiceTask gehörende Aktivität kann mehr als einen Eingangswert und einen Ausgangswert haben. Diese Elemente sind verbunden gemäß der Abhängigkeit es zwischen ihnen gibt, z. B., um die Ausgangswerte zu erzeugen, welche Eingabewerte verwendet werden müssen. Um diese Verbände festzusetzen, werden zwei Elemente zum <ioSpecification>-Element außer die <DataInput>- und <DataOutput>-Element hinzugefügt. Diese Elemente sind: Die <InputSet>, die die Referenzen zu allen von der Aktivität benutzten Eingangsdatenobjekten hat sowie eine Referenz zu der <OutputSet> deren Ausgangsdatenobjekte mit den erwähnten Eingangsdatenobjekten verbunden sind. Im Gegenzug analog, um die gegenseitige Korrespondenz zwischen den *Sets*, die <outputSet> enthält die Referenzen zu den Ausgangsdatenobjekten und eine Referenz zu der *inputSet*, deren Eingangsdatenobjekte mit den oben erwähnten Ausgangsdatenobjekten verbunden sind.

Wenn ein Service-Aufruf durchgeführt wird, sind alle Eingabedaten erhalten, die durch das <SourceRef>-Element im <DataInputAssociation> referenziert sind. Das entsprechende Zuweisungsverfahren, das entweder im <assignment>- oder im <transformation>-Element sich findet wird ausgeführt, um am Ende das Datenobjekt zu bekommen, das durch das <DataInput>-Element referenziert ist, das wiederum in der <ioSpecification> enthalten ist. Später, wenn die Web-Service die Verarbeitung von allen Daten durchführt, wird der interne Ausgabewert erhalten, der im <DataOutputAssociation>-Element gekapselt ist und der durch das <SourceRef> referenziert ist, um den später zu transformieren und als Enderückgabewert zu liefern.

3.1.2 Service Aufrufe in BPEL

Um einen Anruf an einen zum BPEL Prozess externen definierten Service durchzuführen, sollte ein Aufruf durch das <invoke>-Konstrukt ausgeführt werden. Dieses Konstrukt besteht aus Attributen, die zur Durchführung der Kommunikation mit dem Dienst wichtig sind, wie der PartnerLink und PartnerlinkType, die mit einem bestimmten Port-Typ verbuden sind, wo eine benötigte Operation vom Service geleistet wird. Anderseits, andere <invoke> Attribute sind verwendet, entweder um die Variable zu referenzieren, die den Rückgabewert vom Aufruf enthält oder die den Eingabewert enthält, der an den Web-Service als Parameter übergeben wird.

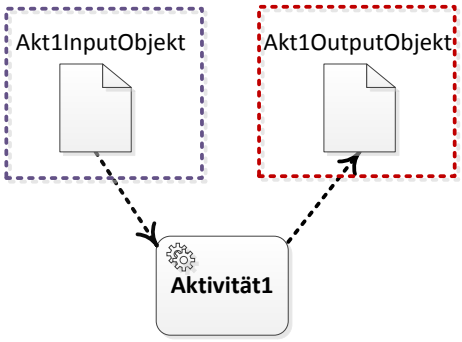
Zunächst, das wichtigste Element vom <invoke>-Konstrukt ist das *operation* Attribut, das auf die Definition der in der importierten WSDL-Datei Operation verweist. Um diese Definition zu erreichen, soll man den Inhalt vom *Porttype* Attribut extrahieren, der dem Namen vom

Porttype im WSDL-File entspricht, in dem diese Operation geleistet ist. Sobald der *Porttype* im WSDL-File gefunden wird, kann man die gekapselte Operationsdefinition suchen und erhalten. Alle diese Elemente oben gezeigt werden in der WSDL definiert. Das <invoke> besitzt zwei zusätzliche Attribute *inputVariable* und *outputVariable* die repräsentieren die Werte, die der Web-Service als Eingabe bekommt und als Rückgabe liefert und die im selben File deklarierte und definierte Variablen referenzieren müssen. Diese deklarierten Variablen haben einen bestimmten Typ, der im WSDL-File auch definiert ist. Dieser Typ ist entweder *type* oder *messagetype*.

Wenn der bestimmte Service durch das <invoke> aufgerufen wird, wird die im <invoke> enthaltene Operation durchgeführt. Die Kommunikation zwischen den beiden PartnerLinks wird eingerichtet und wenn die Operation durchgeführt wird, wird das erhaltene Ergebnis in der Variablen gespeichert, die durch das Attribut ÖutputVariable"referenziert ist.

3.1.3 Transformation einer Servicetask mit Daten

Als erstes sind die Variablen-Namen von den Ursprungsvariablen *SourceRef* im <dataInputAssociation>-Element extrahiert, die die Eingabeparameter repräsentieren, die vom Hauptprozess an die *ServiceTask* übergeben werden. Diese werden in Abbildung 3.2 mit violett gekennzeichnet. Als zweites werden auch die Variablen-Namen von den Zielsvariablen *TargetRef* im <dataOutputAssociation>-Element erhalten, die die Rückgabewerte repräsentieren, die von der *ServiceTask* an den Hauptprozess zurückgegeben werden. In Abbildung 3.2 werden diese mit violett gekennzeichnet. In dieser Abbildung zeigt man die graphische und XML-Darstellung in BPMN von diesen oben genannten Variablen die in BPEL überführt werden. Anschließend werden die Variablen-Namen von der *ioSpecification* Deklaration der BPMN ServiceTask erhalten, die den Eingabe- (DataInput) und Rückgabe-Parametern (DataOutput) entsprechen. Die Eingabe-Parameter werden in in Abbildung 3.3 mit grün gekennzeichnet und die Ausgabe-Parameter werden in Abbildung 3.3 mit orange gekennzeichnet. Wenn diese in der DatenObjekte-Liste vom BPMN-File auch deklariert sind, werden die als Variablen in BPEL zu dem <Variables>-Konstrukt hinzugefügt.

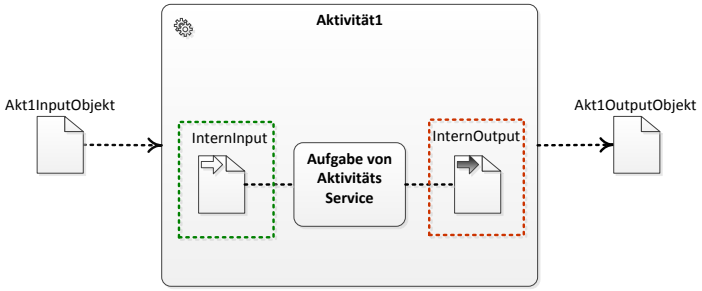



```
<serviceTask name="Aktivität1" id="Act_1">
  ...
  <ioSpecification>
    ......
  </ioSpecification>

  <dataInputAssociation>
    <sourceRef>Akt1InputObjekt</sourceRef>
    <targetRef>...</targetRef>
  </dataInputAssociation>

  <dataOutputAssociation>
    <sourceRef>...</sourceRef>
    <targetRef>Akt1OutputObjekt</targetRef>
  </dataOutputAssociation>
 ...
</serviceTask>
```

Abbildung 3.2: Darstellung in BPMN von verwendeten Variablen in einer ServiceTask




```
<serviceTask name="Aktivität1" id="Act_1">
  ...
  <ioSpecification>
    <dataInput id="InternInput"/>
    <dataOutput id="InternOutput"/>
  </ioSpecification>

  <dataInputAssociation>
    <sourceRef>Akt1InputObjekt</sourceRef>
    <targetRef>InternInput</targetRef>
  </dataInputAssociation>

  <dataOutputAssociation>
    <sourceRef>InternOutput</sourceRef>
    <targetRef>Akt1OutputObjekt</targetRef>
  </dataOutputAssociation>
 ...
</serviceTask>
```

Abbildung 3.3: Darstellung in BPMN von verwendeten internen Variablen in einer Service-Task von [Ley12]

Sobald die Variablen zu der Liste hinzugefügt werden, wird geprüft, ob das <dataInputAssociation>-Element ein <assignments>- oder <transformation>-Element enthält. Wenn der erste der Fall ist, wird den Inhalt von den <from>- und <to>-Tag extrahiert, der danach Anwendungsfallabhängig geparst sein muss. Der Inhalt kann verschiedene Formate haben, entweder einen Pfad der hinweist wo genau der Daten ist und welche *parts* von diesem Daten benutzt

wird, falls dieser Daten eine Nachricht ist, ein Grundtyp z. B. String oder einen Namen einer Variable. Wenn der Inhalt geparst ist, wird es zusammen mit der assignments>-Struktur in eine BPEL-<assign>-Struktur überführt, der die Zuweisungen enthalten wird. Dieses wird durch das <copy>-Element dargestellt, in Abbildung 3.4 zeigen wir ein einfaches Beispiel in dem ein String in eine Variable zugewiesen wird im BPMN <dataInputAssociation>-Element. Wenn das gefundene Element, das <transformation> ist, wird der enthaltene Ausdruck extrahiert und geparst und die entsprechenden Veränderungen durchgeführt.

```
<dataInputAssociation>
  <assignment id="_68_A_1">
      <from>string('2500')</from>
      <to>Number_Orders</to>
  </assignment>
  <sourceRef>Orders_source</sourceRef>
  <targetRef>Number_Orders</targetRef>
</dataInputAssociation>
```

```
<bpel:assign validate="no" name="_68_A_1">
   <bpel:copy>
     <bpel:from>
          2500
     </bpel:from>
     <bpel:to variable="Number_Orders"></bpel:to>
   </bpel:copy>
</bpel:assign>
```

Abbildung 3.4: Beispiel von der Transformation eines BPMN Assignments in ein BPEL assigns

Die aus der Transformation erhaltene *assigns* werden vor (im Fall von <dataInputAssociation>-Element) und nach (im Fall von dataOutputAssociation Element) der entsprechenden Überführung von der ServiceTask gestellt. Abhängig vom Inhalt der Tags <from> und <to> wird die Transformation ändern. Dies bedeutet, dass wenn der Inhalt einem Variablennamen entspricht, wird dieser Name direkt überführt 1:1 wie gezeigt in der Überführung des <to>-Elementes in Abbildung 3.4. Auf der anderen Seite, wenn der Inhalt ein Pfad von einer Variable und ihr Element ist wird es so überführt: In Abbildung 3.5 zeigen wir ein Beispiel in dem der Inhalt vom <to>-Element $getDataObject(Din_10)/parameters/instanceType$ ist. Der erste Wert vom Pfad ist die Funktion, die die definierte BPMN Variable abfragt, die als Parameter übergeben wird, der zweite ist das enthaltene Element und der dritte ist das so zu sagen subElement. Als erstes transformiert man die „getDataObject", „getDataInput“ oder „getDataOutput“ Funktion direkt in $ mit dem Namen der Variablen zusammengekettet mit dem *XPath*. Wenn der Wert im <to>- oder <from>-Element die $String('')$ Funktion ist, soll der enthaltene String direkt als literal transformiert wie gezeigt in Abbildung 3.5. In diesem Beispiel 3.5 ein bestimmter Wert wird in einer Variablen zugewiesen. Wenn es so passiert

benutzt man das <literal>-Konstrukt und drinnen ist das Element mit den gehörenden Teilen eingefügt.

```
<dataInputAssociation>
  <assignment id="_68_A_1">
      <from>string('TMicro')</from>
      <to>bpmn:getDataObject(Din_10/
          parameters/insanceType</to>
  </assignment>
  <sourceRef>Orders_source</sourceRef>
  <targetRef>Din_10</targetRef>
</dataInputAssociation>
```

```
<bpel:assign validate="no" name="_68_A_1">
   <bpel:copy>
     <bpel:from>
       <bpel:literal xml:space="preserve">
           T1Micro
     </bpel:literal>
      </bpel:from>
      <bpel:to>$Din_10/parameters/insanceType</bpel:to>
   </bpel:copy>
</bpel:assign>
```

Abbildung 3.5: Beispiel von der Transformation eines BPMN Assignments in ein BPEL assigns mit <from> und <to> relativ komplexe Pfaden

Danach wird der Inhalt des *Operation* Attributs extrahiert, der die ID der Operation hat. Mit dieser ID, wird es in der Liste von deklarierten Operationen, die im BPMN-File <instance>-Element geschachtelt sind, nach der entsprechenden Operation gesucht. Wenn die Operation gefunden wird, erhaltet man die Schnittstelle <*interface*> in der diese Operation geschachtelt ist. Danach extrahiert man den Inhalt des „ImplementationRef“ Attributs, der verweist auf den Porttype im enthaltenden WSDL-File. Der Porttype-Name und der Operation-Name werden gespeichert, um die später zu dem BPMN-<invoke> *porttype* Attribut bzw. zu dem *operation* Attribut zuzuweisen. Eine Liste von *Porttypes* wird erzeugt mit allen verschiedenen *Porttypes*. Für jeden verschiedenen *Porttype* in der Liste wird einen neuen Partnerlink generiert und zu der <PartnerLinks>-Element eingefügt. Je *PartnerLink* wird so generiert: der Name vom Porttype wird abgefragt, mit dem *plink* String verkettet und dem *name* Attribut vom <partnerLink>-Element zugewiesen. Der Porttype-Name wird auch mit dem *plinkt* String verkettet und dem *partnerLinkType* Attribut zugewiesen. Danach dem andere Attribut *partnerRole* wird der Wert vom operations-Name verkettet mit „plink“ und „partner“ um hinzuweisen die Rolle von der aufgerufenen Service.

Wenn der UnterTyp der Aktivität *Receive* ist, bedeutet das, dass diese Aktivität auf eine Nachricht wartet. Das wird in ein BPEL-<receive>-Konstrukt transformiert. Man nimmt die Attribute die in BPMN-<receiveTask>-Konstrukt sind und überführt sie in BPEL. Der Wert vom Attribut „OperationRef“ wird extrahiert und auf das BPEL-<receive> „operation“ Attribut kopiert, der „Instantiate“ Attributs Zustand (true oder false) wird dem BPEL-<receive> „createInstance“ Attribut zugewiesen (yes bzw. no), und der Wert im „MessageRef“ Attribut wird auf „messageExchange“ kopiert. Diese Referenz weist hin, welche die erwartete Nachricht ist. Zusätzlich wird im „variable“ der Name von der Variable zugewiesen, die den Inhalt der Nachricht speichern wird. Wenn die Struktur dieser Nachricht aus mehrere

Elemente besteht, wird das <fromParts>-Konstrukt drinnen das <receive> eingefügt. Wo jedes in der Nachricht enthaltene Element mit Hilfe des <fromPart> Konstrukt einer Variablen zugewiesen wird. Später wenn eine *Receive* Datenfluss hat, werden die Zuweisungen einfacher sein, da alle „parts“ von der Nachricht, die empfangen wurden, schon in einer Variable sein werden. Dieses hilft wenn die Parts nicht auf dem selben DatenObjekt zugewiesen werden müssen. Das Transformationsverfahren vom *operation* Attribut ist ähnlich dem benutzt, um die ServiceTask zu transformieren 3.1.3, auch die Generierung von Partnerlinks ist ähnlich dem erwähnten Verhalten mit dem Unterschied, dass der erzeugte partnerLink das Attribut *partnerRole* nicht hat, sondern das *myRole* dessen Wert eine Verkettung vom partnerlink-Namen, das String „plink“ und das String „self“ ist.

Wenn der Typ der Aktivität *Send* ist, wird es in ein <invoke>-Konstrukt überführt. Die Attribute werden analog wie mit der *Receive* transformiert, außer dem „Instantiate“, das nicht Teil vom Send ist. Und im Fall von der *ServiceTask* gibt es kein „MessageRef“ Attribut. Schließlich wenn die Aktivität keinen Typ hat, wird sie in eine <empty> Aktivität überführt. Da diese, keine bestimmte Funktion übernimmt.

3.1.4 Aufgaben mit Markern (Schleife, Parallel, Teilprozess oder Kompensation)

Außer der normalen Typisierung kann man die Aufgaben als Schleifen, Mehrfachinstanzen, Kompensationen oder Teilprozesse markieren. Der „Marker“, wie in der Abbildung 3.6 gezeigt, können mit den zugeordneten Typen kombiniert werden um die Aufgabe in ein entsprechendes BPEL-Konstrukt zu überführen.

Abbildung 3.6: Aktivitäten Marker

3.1.4.1 Schleifen

Die Schleifen-Aufgaben werden so lange wiederholt, bis eine definierte Bedingung gilt oder nicht mehr gilt. Es ist eine Art von Repräsentation einer Schleife, aber mit nur einer einzelnen Aktivität. Wenn die Aufgabe einen Schleifen-Marker hat, wird die Überführung diese Aufgabe in einem <while>-Konstrukt geschachtelt und die Bedingung, die mit dem Markern der Aufgabe assoziiert ist, wird als Bedingung dem <while>-Konstrukt zugewiesen.

3.1.4.2 Mehrfachaufgabe

Die Mehrfachaufgaben werden mehrfach instanziiert und können sequentiell oder parallel ausgeführt werden. Wenn dieser Marker in einer Aufgabe identifiziert ist, wird die Überführung der Aufgabe in einem <foreEach>-Konstrukt geschachtelt.

3.1.4.3 Teilprozesse

Andererseits, wenn eine identifizierte Aufgabe ein Teilprozess ist, wird der zu den entsprechenden Aufgaben assoziierte Graph erhalten, durch die rekursive (BpmnProcTree2BpelModelPart) Funktion wird der Graph transformiert und die resultierende Überführung wird in einem <scope>-Konstrukt geschachtelt.

3.1.5 Kompensationen

Diese sind normalerweiser die elementaren Aufgaben, die in einem *Ausnahmefluss* von einer Kompensation sich befinden. Dieser Marker wird in der Überführung der Aktivitäten nur betrachtet wenn die Aktivitäten in einem Kompensations-Ausnahmefluss identifiziert werden. Das wird in dem Abschnitt 3.3.2.4 erläutert.

3.2 Sequenzfluss

Der Sequenzfluss beschreibt die logische Reihenfolge, in der die Flusselemente (Aufgaben, Ereignisse und Gateways) durchgeführt werden sollen. Er repräsentiert den Prozesspfad, über den der Token wandert. Alle Aktivitäten, Gateways und Ereignisse müssen in einer fortlaufenden Folge durch Sequenzflüsse vom Start-Ereignis bis dem End-Ereignis verbunden sein. [Sil11]

Ein Sequenzfluss wird in eine normale gerichtete Kante transformiert, mit dem Sequenzfluss-Element als Inhalt, die Teil vom BPMNProcessTree ist.

3.3 Ereignisse

Die Ereignisse sind Dinge die passieren. Diese beschreiben wie ein Prozess auf ein bestimmtes Signal oder Situation reagieren soll, oder wie der Prozess ein bestimmtes Signal erzeugt, die hindeutet das etwas passiert ist [FR10].

3.3.1 Eingetretene Ereignisse

Diese Ereignisse sind auf einen definierten Auslöser bezogen und werden aktiviert wenn der Auslöser gefeuert wird. Um die Ereignisse überzuführen, werden zuerst die trivialen Komponenten im RPST gesucht. Wenn diese Strukturen gefunden sind, wird der Eingangsknoten überprüft, um zu bestätigen, ob das enthaltene Element vom Typ *Catch-Event* ist. Danach, erhält man die Ereignisdefinition und ihr Typ wird überprüft um später davon abhängig (Nachrichten, Signale, Bedingung, Zeit, Fehler) die entsprechende Transformation durchzuführen.

Im Folgenden werden die verschiedenen unterstützten Fälle erläutert, die vorkommen können:

3.3.1.1 Nachrichten

Diese Definition stellt das Warten auf die Lieferung einer bestimmten Nachricht dar. Wenn das geschehen ist wird die Ausführung von den folgenden Aktivitäten im Pfad fortgesetzt.

Die entsprechende BPEL Struktur mit einem ähnlichen Verhalten dem Nachrichten-Ereignis, ist das <receive>-Konstrukt. Also, wenn ein Ereignis dieses Typs gefunden wird, wird es direkt in ein <receive> transformiert und zusätzlich werden die *Nachrichten*, *Port* und *Porttype* Eigenschaften hinzugefügt.

3.3.1.2 Zeit

Diese Definition präsentiert eine bestimmte Zeitbedingung die erfüllt werden soll, entweder eine spezifische Dauer oder die Erwartung eines Datums bzw. Zeit, um die Ausführung dieses Flusses fortzusetzen.

Dieses Ereignis wird direkt in eine BPEL-Struktur überführt, die am besten sich eignet: Das <Wait>-Konstrukt. Die Bedingung wird von der *Definition* extrahiert und ausgewertet. Wenn die erhaltene Bedingung den Typ *duration* hat, wird die in eine *for* Bedingung transformiert, die eine bestimmte Wartezeit repräsentiert. Auf der anderen Seite, wenn die erhaltene Bedingung vom Typ *Datum* ist, wird die in eine *setUntil* Bedingung transformiert, die ein Datum repräsentiert, in dem die Ausführung fortsetzten muss. Wenn die Bedingung erstellt ist, wird sie zum <wait>-Konstrukt hinzugefügt.

3.3.1.3 Bedingung

(a) Unterbrechende **(b)** Nicht-unterbrechende

Abbildung 3.7: Angehefteten Ereignisse

Diese Definition repräsentiert eine bestimmte Bedingung die erfüllt werden soll, um das Prozess fortzusetzen. Dieses Ereignis kann als Zyklus verstanden werden, das wartet, bis eine Bedingung erfüllt ist. Das kann in BPEL mit einem <repeat>-Konstrukt repräsentiert, das als Bedingung die verneinte Version von der originale Bedingung hat. So dass nur wenn die Bedingung erfüllt ist, setzt die Ausführung fort

3.3.1.4 Links

Das Linkereignis ist ein Werkzeug, das keine besondere Bedeutung hat, es hat als Zweck die Darstellung eines Prozessdiagramms zu erleichtern. Zwei zusammengehörige Links, ein ausgelöstes und ein eingetretenes, die dieselben Namen haben, funktionieren praktisch wie eine Kante. Diese Ereignisse werden unterschiedlich behandelt. Wenn der erste Schritt der Transformation durgeführt wird (Überführung in einen Graph), wird für jeden Endknoten, dessen enthaltenes Element ein Link Ereignis ist, sein entsprechender Startknoten gesucht, um die beide mit einer Kante zu verknüpfen. Danach, wenn die Überführung in BPEL gemacht wird, wird jedes Link Ereignis nicht weiter Verarbeitet sonder geskippt.

3.3.2 Angehefteten Ereignisse

Ein angeheftetes Ereignis wird durch ein Ereignis am Rand einer Aktivität dargestellt, wie in Abbildung 3.7a gezeigt. Ein Angeheftetes Ereignis hat nicht als Funktion, dass diese Aktivität auf etwas für eine bestimmte Zeit wartet, sondern dass während die Aktivität ausführt, der Prozess ist bereit die Behandlung dieses bestimmten Ereignisses durchzuführen, falls es auftritt.

Das angeheftete Ereignis hat ein verschiedenes Verhalten abhängig vom Rahmen des Ereignisses. Wenn der Rahmen durchgängig ist, bedeutet es dass wenn das Ereignis auftritt, wird

die Ausführung von der Aktivität abgebrochen, an der das Ereignis angeheftet ist und der zum Ereignis gehörende Fluss ausgeführt.

Vor der Durchführung der Transformation, ist es wichtig, zu erklären, dass beim ersten Schritt der Überführung der XML-Datei in den ProcessTree es einen Unterschied zwischen Hauptfluss und Ausnahme-Fluss gibt. Man kann den Unterschied visuell erkennen, Abbildung 3.8 zeigt ihn. Diese Unterscheidung wird durch die Hinzufügung eines Flags zu jedem Knoten erreicht, das hinweist zu welchem Flusstyp der Knoten gehört. Am Anfang ist das Flag *MainFlow* von allen Knoten im Diagramm auf falsch gesetzt.

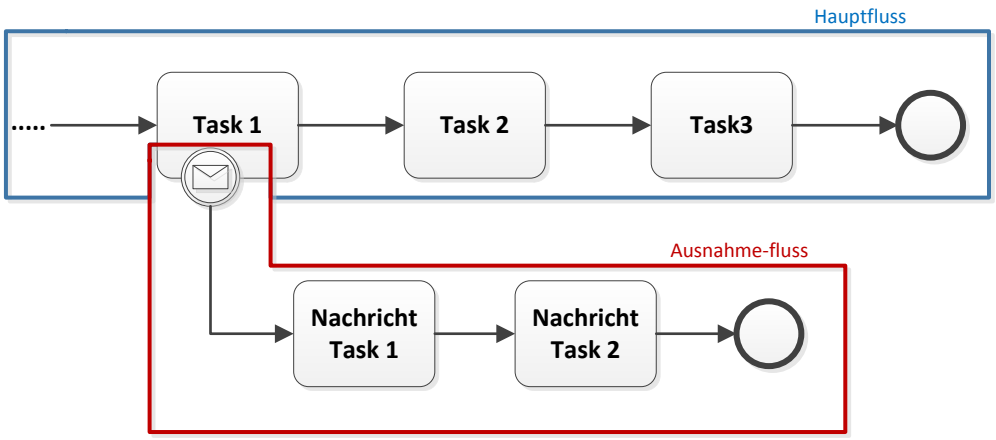


Abbildung 3.8: Visueller Unterschied zwischen Haupt- und Ausnahme-fluss

Beim ersten Schritt der Transformation, in dem der Graph vom Prozess Startknoten aus traversiert wird, wird die *MainFlow* Eigenschaft jedes Knotens auf *true* gesetzt. Sobald der dem entsprechende Hauptfluss Graph erzeugt wird, werden die im XML erhaltenen angehefteten Ereignisse gesucht. Für jedes wird die Aktivität erhalten, die mit dem Ereignis verknüpft ist und der Weg, der mit dem Ereignis anfängt, wird traversiert. Jeder Knoten der besucht ist, wird geprüft um zu bestätigen ob den zum Hauptfluss gehört. Im Falle, der Knoten nicht dem Hauptfluss gehört, wird dieser zum entsprechendem Ausnahme-Fluss Teilbaum hinzugefügt und der Weg weiter durchgelaufen. Sonst, wenn der Knoten dem Hauptfluss gehört, wird er zum Teilbaum hinzugefügt aber der Weg wird nicht mehr durchgelaufen, weil der *Ausnahme-Fluss* mit diesem Knoten endet. In diesem Fall wird es dazu verwendet, um den Kontrolfluss auf diese „Ende"Task zu leiten. Im letzteren Fall wird der Knoten das *MainFlow* Flag nicht auf false gesetzt, da alle Knoten standardmäßig auf true gesetzt sind. Diese Klassifizierung von Knoten wird sehr hilfreich sein, wenn die den entsprechenden *Ausnahme-Flüsse* Graphen analysiert und in BPEL überführt werden.

Sobald der Graph herstellt wird, wird es zu der Liste der *Ausnahme-Fluss* Graphen der assoziierte Aktivität hinzugefügt. Wenn alle Graphen zur Liste der *Ausnahme-Flüsse* der Aktivität hinzugefügt werden, vervollständigt sich der erste Schritt der Transformation.

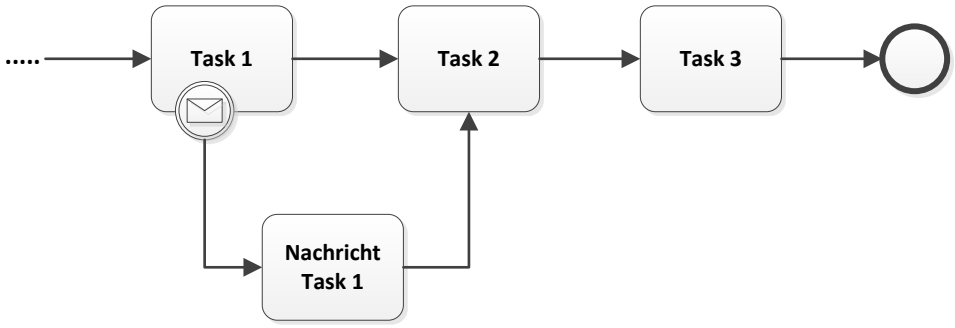
Die angehefteten Ereignisse können als Handlers verstanden werden, die eine spezifische Aktivität durchführen, wenn ein bestimmtes Ereignis auftritt. Ein Ereignis kann unterbrechende oder nicht-unterbrechende sein. Wenn das Ereignis unterbrechend ist und es auftritt, wird die Aktivität abgebrochen die zurzeit durchführte zusätzlich mit allen folgenden Aktivitäten während die zum aktivierten Handler gehörenden Aktivitäten durgeführt werden. Wenn das Ereignis nicht-unterbrechend ist, sollen die mit dem Ereignis assoziierten Aktivitäten parallel zu den zum Hauptfluss gehörenden Aktivitäten ausgeführt werden.

Die nächsten Schritte der Transformation bestehen in der Überprüfung jeder Aktivität, um zu finden welche ein *Ausnahme-Fluss* haben, in welchem Fall sie schrittweise erhalten werden und für jeden Ausnahmefluss der entsprechende RPST berechnet wird, um analysiert zu werden. Mit dem RPST schon berechnet, nimmt man das erhaltene Element des Startknotens und prüft man den Typ des Ereignisses, um hinzuweißen welche Handler zum <Scope>-Konstrukt hinzugefügt wird.

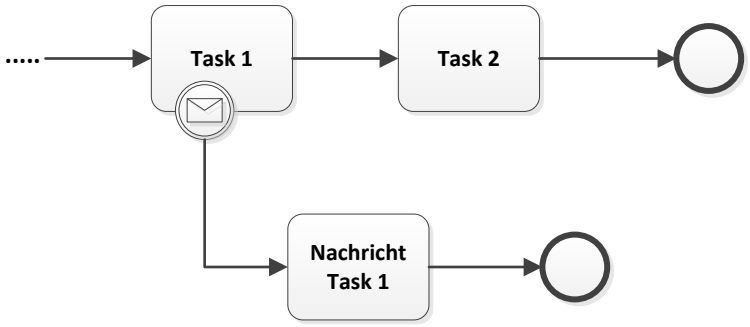
Die Überführung des Ereignisses ändert abhängig vom Typ, und die Typen von Ereignis die behandelt und transformiert werden sind: Nachricht, Zeit, Fehler, Kompensation und Mehrfach.

3.3.2.1 Nachrichten

Wenn ein angeheftetes Nachricht-Ereignis mit einer Aktivität verknüpft ist, bedeutet dieses, dass während der Aktivität ausführt eine bestimmte Nachricht erwartet wird. Wenn diese ankommt, wird abhängig von dem Typ des Ereignisses, die Ausführung der laufenden Aktivität unterbrochen oder nicht. Das gibt zwei Verhaltensmöglichkeiten:



(a) unterbrechendes angeheftetes Nachricht-ereignis, das Verbindung mit dem Hauptfluss hat



(b) Unterbrechendes angeheftetes Nachricht-ereignis ohne Verbindung mit dem Hauptfluss

Abbildung 3.9: Unterbrechende angeheftete Nachricht-Ereignisse

Die erste ist in Abbildung 3.9a dargestellt. Die laufende Aktivität *Task*1 wird untergebrochen und die mit dem Ereignis assoziierten Aufgaben ausgeführt und dann, wenn das Ende des *Ausnahme-Flusses* mit dem Hauptfluss durch eine Kante verbunden ist, werden die dem Ausnahme-Flusses Endeknoten folgenden Aktivitäten ausgeführt. Ansonsten, wie in Abbildung 3.9b gezeigt, werden nur die Aktivitäten durchgeführt, die mit dem Ereignis assoziiert sind, und danach wird die Ausführung beendet. Die Zweite Möglichkeit ist in Abbildung 3.10 dargestellt. In diesem Beispiel wird die laufende Aktivität $Aufgabe1$ nicht unterbrochen und die Aufgaben vom Ausnahme-Fluss werden parallel mit den Aktivitäten des Hauptflusses ausgeführt.

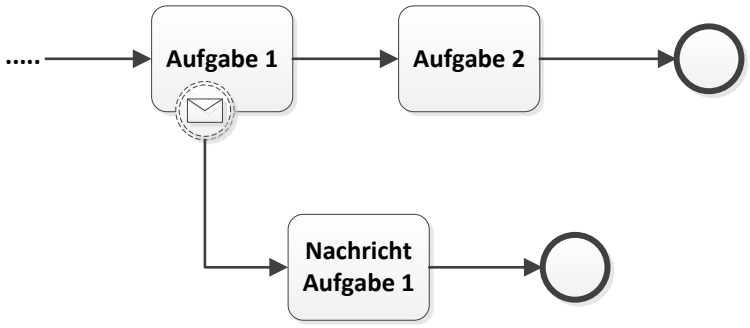


Abbildung 3.10: Nicht-unterbrechendes angeheftetes Nachricht-ereignis

Um dieses in BPEL zu übersetzen, gibt es keine direkte Struktur die man benutzen kann, sollte man kombinierte Strukturen verwenden, um dieses Verhalten zu repräsentieren:

Zuerst, soll ein <EventHandler>-Konstrukt erzeugt werden, es ist die einzelne Methode die BPEL anbietet um explizit hinzuweisen, welche Aktivitäten ausgeführt werden sollen wenn bestimmte Ereignisse auftreten. Ein <OnMessage>-Element wird erstellt, mit dem man durch die Partnerlink und Operation Attribute hindeutet mit welchem Nachrichtempfang das Ereignis assoziiert ist und in dem Element werden die Aktivitäten geschachtelt sein, die ausgeführt werden müssen.

Falls das Ereignis den Typ „unterbrechend“ hat, bedeutet es semantisch, dass wenn das erwartete Ereignis auftritt, die Ausführung des Hauptflusses abgebrochen wird, so dass der „Ausnahmefluss“ dieses Ereignisses durchführt. Das BPEL Konstrukt, das erlaubt, den Hauptfluss abzubrechen, ist das <FaultHandlers>-Konstrukt, das die Ausführung stoppt wenn ein <throw>-Konstrukt durchführt, das kann sich entweder in einem bestimmten *Event Handler* befinden, der ein Ereignis behandelt, oder als eine Anweisung im BPEL-Code sein. Dieses Konstrukt hilft das Verhalten von BPMN-unterbrechenden-Ereignissen zu modellieren.

Um dieses zu erreichen soll man die folgenden Schritte vorführen: Im <OnEvent>-Element wo die Aktivitäten definiert sind, fügt man eine <throw> Aktivität, die einen bestimmten Fehler wirft. Der Name dieses Fehlers ist derselbe wie der Name des <Catch>, der dieses Ereignis behandeln wird. Die Funktionalität dieses <throw> ist, den Ausführungsfluss zum „Fault Handler“ umzuleiten, damit die Ausführung des Hauptflusses abgebrochen wird.

Abgesehen von diesen beiden möglichen Fälle, gibt es auch die Fälle, in denen der *Ausnahmefluss* eines Ereignisses mit einem Sequenzfluss enden, der mit der Hauptfluss des Prozesses sich verbindet. Diese Ereignisse werden wir als *beeinflussende* erkennen. Die andere *Ausnahmenflüsse* die enden und keine Verbindung mit dem Hauptfluss haben (diese haben

als letztes Element ein End-Ereignis) werden wir als *nicht-beeinflussende* erkennen. Es ist wichtig anzumerken, dass die angehefteten unterbrechenden Nachricht-Ereignisse, die in dieser Arbeit behandelt und berücksichtigt wurden, müssen in einer Aktivität sein, die zur Hauptprozesses Sequenz gehört und nicht in einem Bond enthalten sind. Weil in solche Fälle wie der in Abbildung 3.11 dargestellt, Verklemmungen vorkommen könnten. In diesem Fall sollen beide Aktivitäten *Task*1 und *Task*2 ausgeführt werden, aber wenn das angeheftete Nachricht-Ereignis von *Task*1 auftritt, wird die Ausführung von diesem Zweig abgebrochen und eine Verklemmung vorkommen, da das Join-parallele-Gateway auf den Token des Zweigs *a* wartet.

Im Falle, dass das angeheftete Ereignis nicht unterbrechend und nicht beeinflussend ist, ist die Transformation anders:

Die vorgeschlagene Lösung ist: Ein <Scope> wird erzeugt und drinnen wird die Überführung entsprechend der Aufgabe eingefügt. Ein <EventHandler>-Element wird generiert zusätzlich mit einem <OnEvent> das im <EventHandler> geschachtelt sein wird. Die Aktivität dieses <OnEvent> wird einen Aufruf einer „Invoke“ sein, der auf eine Operation im WSDL-File verweist, die definiert werden muss. Diese Operation entspricht den Aktivitäten die zum *Ausnahmefluss* gehören. Wenn die im <scope> enthaltene Aufgabe schon ausgefürt wird, fährt die Ausführung des Rests der Aufgaben vom Hauptfluss for zusätzlich mit der aufgerufene *Invoke*.

Anderseits, wenn das angeheftete Ereignis beeinflussend ist, ist die Transformation ähnlich der oben erklärten Transformation. Der Unterschied ist, dass eine *Receive* nach der Aktivität, die als Endeknoten mit dem Ausnahmefluss verbunden ist, hinzugefügt werden muss. Diese Receive wird auf die Nachricht warten, die hinweist wann die Aktivitäten vom Ausnahmefluss, in diesem Fall die *Invoke*, fertig sind.

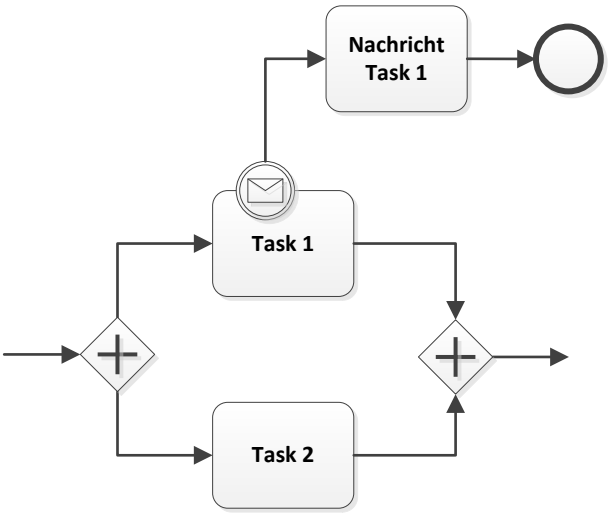
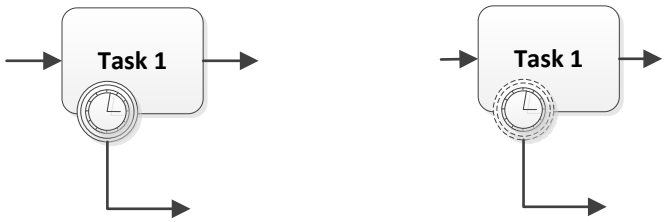


Abbildung 3.11: Nicht behandelte Fall von angehefteten unterbrechenden Ereignissen

Um diese Ereignisse zu überführen, dessen Fluss keine Verbindung mit dem Hauptfluss haben, werden nur zum Fault Handler die mit dem Ereignis assoziierten Aktivitäten hinzugefügt, während in anderen Fällen wird auch einen <Source> Link am Ende der Aktivitäten des <FaultHandler> eingefügt, der den Fluss auch nach dem Hauptfluss umleitet. Man macht das durch die Hinzufügung eines <Target>Links zu der im Hauptfluss erhalten Aktivität. Diese Überführung wurde so, durchgeführt:

Nachdem die *Polygon*2.3 Strukturen identifiziert werden, d. h. Sequenzen die als Startelement ein Zwischenereignis haben, das auch an einer Aktivität oder Teilprozess angeheftet ist. Prüft man ob die dem entsprechenden „Ausnahmefluss“ Polygon, mit einer Verbindung mit dem Hauptfluss endet. Danach sind Sie die ID der Aktivität/Teillprozess mit dem der „Ausnahmefluss“ sich verbindet, ein Source-Link wird erstellt und am Ende der <FaultHandler> Aktivitäten hinzugefügt. Eine Hash-Tabelle wird erstellt, die als Key die Id der Aktivität haben wird, die zum Hauptfluss gehört und auf die vom Ausnahmefluss geleitet wurde und als Element wird die Hashtabelle eine Menge von Links erhalten, die mit der Aktivität als *TargetLinks* assoziiert sein müssen. Dies ist dazu verwendet um den Kontrolfluss im BPEL File hinzudeuten. Es wird für alle angehefteten Ereignisse der Aktivität gemacht. Wichtig hervorzuheben ist, dass wenn eine Aktivität angeheftete Ereignisse hat, sollte diese Aktivität in ein <scope>-Konstrukt überführt, das alle die entsprechende „Handlers“ enthalten wird. Solltet man noch eine Aktivität oder Teilprozess im Hauptfluss finden, der angeheftete Ereignisse hat, ist die Behandlung die gleiche.



(a) Unterbrechendes angeheftetes Nachricht-ereignis das Verbindung mit dem Hauptfluss hat

(b) Unterbrechendes angeheftetes Nachricht-ereignis ohne Verbindung mit dem Hauptfluss

Abbildung 3.12: Angehefteten Zeit-Ereignisse

Wenn alle „Ausnahmeflüsse“ analysiert werden und die Links zu der Tabelle hinzugefügt, gehen wir weiterhin die Analyse der Strukturen des Hauptprozesses. In diesem Prozess der Analyse wird für jede Aktivität geprüft, ob die ID Schlüssel eines in der Hash existierendes Elementes ist, wenn ja, werden die in dem Hasheintrag <Target>Links während der Überführung dieser Aktivität am Ende des Konstrukts der entsprechende Transformation hinzugefügt. Und danach wird der Eintrag von der Hash gelöscht.

3.3.2.2 Zeit

Wenn solche angeheftete Ereignisse mit einer Aktivität verknüpft sind, wird es erwartet während der Ausführung der Tätigkeit, bis eine bestimmte Zeitbedingung sich erfüllt, um den „Ausnahmefluss“ durchzuführen. Wenn die Aktivität vollständig durchgeführt wird, vor der Zeitbedingung vom Ereignis erfüllt werden kann, der normale Hauptfluss vom Prozess läuft weiter. Sonst, falls die Aktivität unterbrechend ist, wird die Aktivität unterbrochen.

Zunächst werden die Polygons identifiziert, die als Startknoten ein Zeit angeheftetes Ereignis haben, das an eine Aktivität angeheftet ist. Es wird geprüft, ob das Ereignis unterbrechend ist oder nicht, dann wird es auch geprüft ob der Fluss, der zum Ereignis gehört, in den Hauptfluss reinkommt oder außer dem Hauptfluss bleibt.

Die Struktur, im Falle sie unterbrechend ist, wird gleich wie im vorherigen „Nachrichten“ Fall 3.3.2.1 überführt, mit dem Unterschied, dass im Inneren des <EventHandler>-Konstrukts ein <OnAlarm>-Element statt ein <OnEvent>-Element hinzugefügt wird. Der Rest der Transformation wird gleich behandelt.

3.3.2.3 Fehler

Auf der anderen Seite, wenn der Typ des angehefteten Ereignisses „Fehler“ ist, wird die Aktivität bereit, einen gewissen und möglichen Fehler zu erwarten, der Falls auftritt, wird immer die Aktivität unterbrochen. Es gibt keine Möglichkeit mit diesem Ereignis die Haupt-Flussausführung nicht zu unterbrechen und den mit dem Ereignis assoziierten *Aunahmefluss* gleichzeitig mit dem Hauptfluss laufen zu lassen.

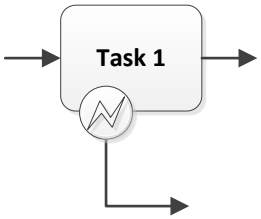


Abbildung 3.13: Angeheftetes Fehler-ereignis

In diesem Fall wird die Überführung leicht verändert, da ein <EventHandler> zum der entsprechenden Aktivität <Scope>-Konstrukt nicht hinzugefügt wird, sondern direkt ein <FaultHandler> der den erwartete Fehler behandelt.

Falls der Ausnahmefluss in den Hauptfluss reinkommt, wird einen Link am Ende des <FaultHandler>-Konstrukts eingefügt, der die letzte Aktivität des „Ausnahmefluss“ mit der Aktivität des Hauptflusses verbindet. Die Verwaltung und Zuteilung der Links wird mithilfe des im Abschnitt 3.3.2.1 vorherigen beschriebenen hashtable durchgeführt.

3.3.2.4 Kompensation

Dieses Ereignis geht darum, dass in einigen Fällen manche Aktivitäten rückgängig gemacht werden sollen, die schon ausgeführt wurden. In Abbildung 3.14 zeigen wir wie dieses Ereignis graphisch dargestellt wird.

Wenn die Kompensationen von BPMN in die Graphen-Darstellung überführt werden, ist das Identifizierungsverfahren der Aktivitäten, die angeheftete Kompensation-Ereignisse haben, ähnlich allen anderen angehefteten Ereignissen gemacht. Der einzige Unterschied ist, dass um den Baum zu erstellen, der den Fluss der Kompensation repräsentiert, soll man nach das angeheftete Ereignis gefunden wird, in der Liste der Assoziationen die Assoziation suchen, die als Quellenelement das gefundene Ereignis hat. Danach wird das Senkenelement von der Assoziation erhalten, jedes folgende Element besucht und der Graph von diesem

Senkenelement aus erzeugt. Danach wird dieser Graph zu der Liste den angehefteten Ereignissen der Aktivität hinzugefügt.

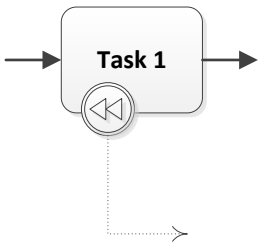


Abbildung 3.14: Angeheftetes Kompensation-ereignis

Dann wird der nächste Schritt des nach-BPEL-Überführungsverfahrens viel einfacher und direkter sein. Der Graph der Ausnahme-fluss wird erhalten und wenn der Graph mit einem Kompensation angehefteten Ereignis startet, wird die Aktivität erhalten, die mit dem Kompensation-ereignis assoziiert ist, und ein <Scope>-Konstrukt wird erstellt indem der <compensationHandler> und seine Aufgaben einbezogen sind.

Da dieses Ereignis verwendet wird, um eine Aktivität oder ein Teilprozess zu kompensieren, das schon durchgeführt wurde, d. h. eine Aufgabe rückgängig zu machen, an der das Ereignis angeheftet ist. Dieses bedeutet, dass einmal die Kompensationsaufgaben durchgeführt werden, soll der Zustand zurück zum Punkt gehen, indem die Aktivität, an der das Ereignis angeheftet ist, noch nicht durchgeführt wurde. Und der Ausführung-fluss soll weiter von der Aufgabe aus laufen, die nach der kompensierten Aktivität in der Reihenfolge steht. Diese Ereignisse sind immer unterbrechend und der entsprechende Fluss endet nicht im Hauptfluss. Die Überführung ist immer die gleich, da es keine andere Variante gibt.

3.3.2.5 Mehrfach

Dieses angeheftete Ereignis besteht in der Zusammenfügung von mehreren Ereignissen auf einem einzigen Ereignis, aber davon wird nur eine behandelt werden. Wenn mehr als ein Ereignis auftreten, wird das erste, das auftritt, behandelt.

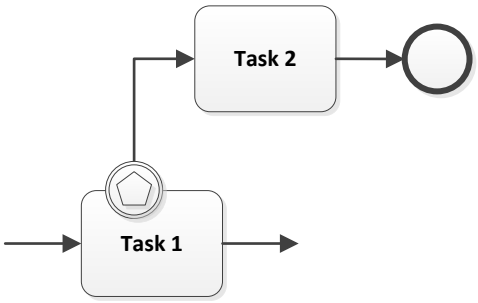


Abbildung 3.15: Angeheftetes Mehrfach-ereignis

Im Falle dieses Ereignisses wird das Standard-Identifizierungsverfahren, erläutert am Anfang des Abschnitts 3.3.2, für andere Ereignisse durchgeführt. Was ändert, ist die BPEL Überführung: Alle Ereignisse-bedingungen werden aus dem Ereignis Definition erhalten und jede Bedingung wird in ein <onEvent>-Konstrukt transformiert und in einem <EventHandlers>-Konstrukt geschachtelt. Falls das Ereignis unterbrechend ist, wird ein <Catch>-Element erstellt das im <FaultHandlers>-Konstrukt gekapselt wird. Danach für jedes erstellte <onEvent>-Konstrukt wird ein entsprechendes <throw>-Element erzeugt und im dem <onEvent>-Element geschachtelt, das denselben Namen vom <Catch> hat und den Fluss auf den FaultHandler umleitet. Das <FaultHandler>-Konstrukt wird die Funktionalität vom Abbrechen des Hauptausführung-flusses übernehmen.

3.3.3 Teilprozesse Handlers

Diese sind *Ausnahme-flüsse*, die in Teilprozesse eingekapselt sind. Sie haben semantisch dieselbe Funktionalität wie die angehefteten Ereignisse. Mit dem Unterschied, dass sie entlang der gesamten Länge des Teilprozesses ein bestimmtes Ereignis erwarten.

Die Identifizierung dieser Ereignisse unterscheidet sich bezüglich welcher Ereignisse drinnen enthalten sind. Um diese Ereignisse zu erkennen, sucht man die Teilprozesse, die auch in einem Teilprozess gekapselt sind, wie gezeigt in Abbildung 3.16, und die Handlers enthalten d. h. der gesuchte Teilprozess soll das Attribut „triggeredByEvents“ auf true gesetzt haben. Dieses Flag identifiziert alle Teilprozesse die nur Aufgaben durchführen wenn ein Ereignis auftritt. Außerdem kann man das auch visuell im BPMN sehen, da der Rahmen des TeilProzesses nicht durchgehend ist.

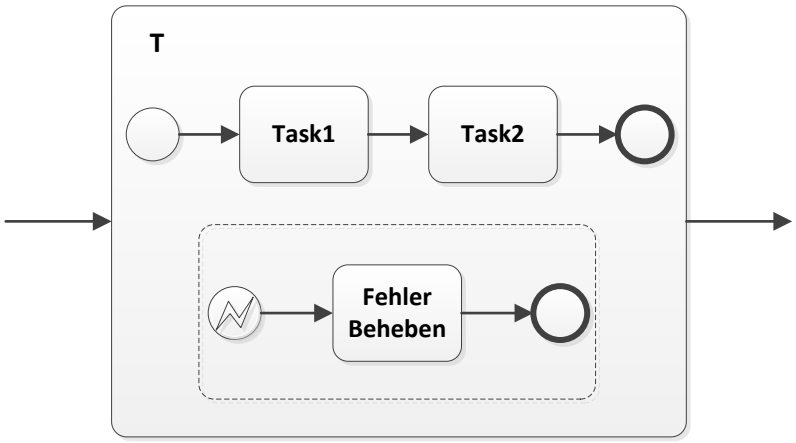


Abbildung 3.16: Beispiel von einem Teilprozess T mit einem enthaltenen Handler

Sobald der Teilprozess gefunden ist, wird dieser weiter analysiert. Das Startevent-Element wird aus dem Teilprozess erhalten, das der Anfang des „Ausnahmeflusses“ und das das erwartete Ereignis repräsentiert. Dieser ganze Fluss wird durchgelaufen und ein Graph wird erzeugt. Der Graph wird zu der Liste des angehefteten Ereignisses des Großelters hinzugefügt, da das Verhalten dasselbe ist als ob das Ereignis angeheftet wäre. Später bei der Überführung dieses Teilprozesses in BPEL, wird die Liste der angehefteten Ereignisse durchgelaufen und zum Teilprozesse-scope hinzugefügt, wie oben im Abschnitt 3.3.2 erläutert.

3.4 Gateways

Wenn man in einem Geschäftsprozess den Fluss steuern will, braucht man Gateways, um explizit hinzuweisen welcher ausgehende Sequenzfluss ausgewählt werden soll. Wenn mehr als ein Sequenzfluss ausgeführt werden soll, beschreibt das Gateway auch wie dieses durchgeführt wird. Das Gateway ist quasi der Punkt, an dem man entscheidet, was als Nächstes zu tun ist.

Es gibt zwei Arten von Komponenten, zu denen die beteiligten Gateways gehören:

3.4.1 Strukturierte

Diese Kategorie besteht aus allen Komponenten die ein Split-Gateway als Eingangsknoten und ein Join-Gateway als Ausgangsknoten haben und keine andere Gateways haben. Diese können abhängig der Typ der Gateways weiter klassifiziert werden.

3.4.1.1 Datenbasiertes exklusives Gateway (XOR)

Die Funktionalität dieses Gateways ist, durch einen Ausdruck von verarbeiteten Daten zu evaluieren, welche von den ausgehenden Flüsse aktiviert werden um die in der Instanz später auszuführen. Das Gateway verfügt über zwei verschiedene grafische Darstellungen: Eine ist ein Diamant Symbol mit einem "Xïm Inneren, und die andere hat kein Symbol drinnen.

Das Transformationsverfahren von diesen Strukturen besteht in die Analyse vom RPST auf der Suche nach “Bond” Muster, die ein XOR Gateway als Eingangsknoten und ein XOR Gateway als Ausgangsknoten haben.

Begriff (Bond): Ein Bond ist eine Komponente die 2 Grenzeknoten hat (Start) u und (Ende) v und die zumindest 2 Pfade oder Sequenzen von Komponenten von u nach v hat. [VVK08]

Nachdem die Komponente durch die vorherige Methode erkannt wird, analysiert man alle Kinder. Für jedes Kind mit „Polygon“ Typ sucht man den ausgehenden Sequenzfluss, der aus dem Gateway geht und das “Polygon” anfängt. Die Bedingung wird extrahiert, danach, wenn dieses Polygon, das erste Kind ist, werden die Aktivitäten transformiert, die zu dem Polygon gehören und direkt zu dem <if>-Konstrukt als Anweisungen hinzugefügt werden. Das <if> wird in einem <else-if>-Konstrukt geschachtelt. Im Falle der enthaltene Polygon nicht das erste ist, sondern ein nachfolgendes, wird die Überführung der Aktivitäten dieses Polygons als Anweisungen im <else-if>-Konstrukt geschachtelt, oder wird zu den Anweisungen des <else>-Konstrukts hinzugefügt, wenn der entsprechende Sequenzfluss der das Polygon anfängt keine Bedingung hat und die Eigenschaft „default“ auf true gesetzt wird.

Stufenweise wird jedes Bedingungskonstrukt, entweder if, else oder else-if, mit den entsprechenden Aufgaben zu dem Haupt <if>-Konstrukt hinzugefügt.

Abbildung 3.17: Mögliche Darstellungen vom Exklusiven Gateway

3.4.1.2 Paralleles Gateway (AND)

Dieses Gateway spezifiziert, dass alle die ausgehenden Sequenzflüsse sollen Parallel und unbedingt ausgeführt werden. Jeder ausgehende Pfad repräsentiert einen konkurrierenden Faden der mit anderen in Zeit überlappen. Es wird durch einen Diamant dargestellt. [Sil11]

Das einzelne BPEL-konstrukt, das erlaubt, eine Parallele Ausführung von Aktivitäten hinzudeuten ist das <flow>. Dieses übernimmt die Synchronisation der Aktivitäten, d. h. nach alle Zweige ausgeführt werden, führt der Rest aus. Um die Strukturen zu transformieren, die aus ausgeglichenen Paaren von Gateways gebildet sind, identifiziert man die "Bond" Muster, die ein AND Gateway als Eingangsknoten und ein AND Gateway als Ausgangsknoten haben.

Sobald die Komponente identifiziert wird, sind die Kinder analysiert. Jedes "Polygon" Kind wird transformiert und die erhaltene Sequenz von Aktivitäten wird zu dem <flow>-Konstrukt hinzugefügt ohne bestimmte Reihenfolge, da alle parallel ausführen.

3.4.1.3 Inklusives Gateway

Es erlaubt eine Und-Oder Situation zu modellieren, bei der entweder einen, mehrere oder alle ausgehenden Pfade gleichzeitig durchlaufen.

Zuerst werden die "Bond" Komponenten identifiziert, die ein inklusives Gateway als Eingangsknoten und ein inklusives Gateway als Ausgangsknoten haben. Da das inklusive Gateway so funktioniert, dass die true-evaluierte ausgehende Kanten parallel ausführen, wird alles in einem <flow>-Konstrukt eingebettet. Danach, für jeden ausgehenden Sequenzfluss wird die Bedingung extrahiert und die assoziierte Sequenz von Aktivitäten erhalten.

Als nächstes, um den Fluss zu steuern, werden Links genutzt, durch die für jede erfüllte Bedingung den Fluss in den entsprechenden Aktivitäten-Block gelenkt wird. Alle diese Aktivitäten-blöcke sind in dem Haupt<flow>-Konstrukt geschachtelt.

Am Ende jedes dem Polygon-Konstrukt Blocks, wird einen Link hinzugefügt, der den Fluss nach dem Ende des Haupt-<Flow> umlenkt.

3.4.1.4 Ereignisbasiertes Gateway

Dieses Gateway leitet den Fluss nicht bezogen auf Daten, sondern abhängig davon, welches erwartete Ereignis zuerst stattfindet. Es handelt sich um eine XOR-Semantik. Das erste Ereignis, das eintritt, wird die dem Ereignis nachfolgende Sequenz durchführen. Das BPEL-Konstrukt das ein ähnliches Verhalten dem BPMN Ereignisbasierten Gateway hat, ist das <pick>. Das <pick>-Konstrukt erlaubt, auf mehrere Nachrichten zu warten, die auf unterschiedliche Weise behandeln werden können. Auch unterstützt das <pick> die Erwartung von Zeitereignissen. Mit diesen Gateways kann nur eine mögliche *well-structured* von BPMN unterstützte Struktur gebildet werden. Um diese Komponenten zu

überführen, werden die “Bond” Muster identifiziert, die ein Ereignisbasiertes Gateway als Eingangsknoten haben und ein XOR Gateway als Ausgangsknoten haben. Danach erhält man die Kinder von der „Bond“ Komponente, und für jedes Kind wird das enthaltene Element von dem Gateway nachfolgenden Knoten überprüft. Sobald dieses Element überprüft wird, wird es abhängig vom Typ vom Ereignis transformiert.

3.4.2 Teilweise strukturierte- und unstrukturierte Strukturen

3.4.2.1 Teilweise strukturierte

Diese Kategorie besteht aus alle Komponenten die nicht zusammengehörige Gateways haben und umstrukturiert werden sollen, um ihre Analyse, Unterteilung in Komponenten und Überführung zu erleichtern. Wenn diese Komponenten nicht ausgeglichene Gateways haben, d. h. dass sie ungerade Nummer von Gateways haben, haben sie die Eigenschaft das Ausgangs- oder Eingangsgateway mit einem „Bond“ Kind zu teilen. Das Umstrukturierungsverfahren dieser Komponenten wird durch die neue Funktion „restructureQuasi“ durchgeführt. Diese Funktion traversiert den RPST vom *Workflow Tree* und sucht drauf rekursiv alle „Bond“ Komponenten. Zuerst identifiziert sie nur die „Bond“ Komponenten die ein „Bond“ Elter haben, für jede dieser Komponenten wird es überprüft ob sie das Eingangs- oder Ausgangsgateway mit ihr Elter teilen.

Wenn das „Kind-Bond“ *B* und das „Elter-Bond“ *C* den gleichen Eingangsknoten haben, wie in Ausbildung 3.18, muss das Gateway auf zwei gespaltet werden, um die Anzahl von den Gateways auszugleichen:

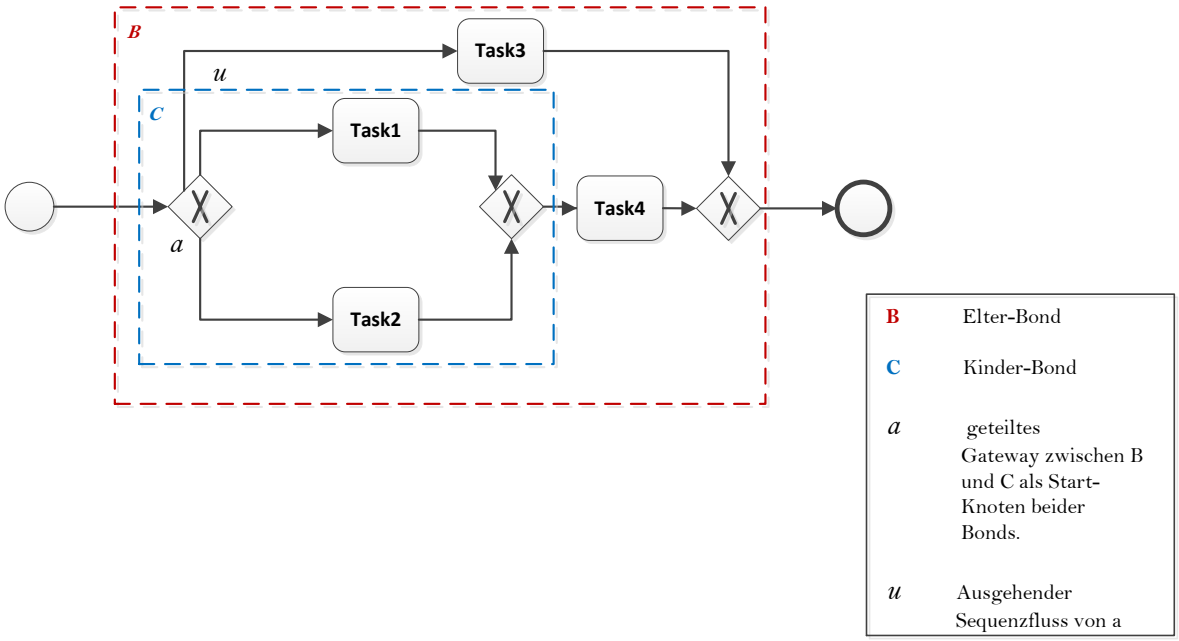


Abbildung 3.18: Beispiel von einer teilweise strukturierte Komponente

Ein neues Gateway a' mit demselben Typ vom geteilten Gateway (parallel oder exklusiv) wird erstellt, eine neue v Kante vom neuen Gateway a' nach dem alten Gateway a wird eingefügt. Dann für jede ausgehende Kante u vom a, die zum „Elter-Bond“ B gehören und nicht zum „Bond“ C, wird den Senkenknoten erhalten, eine neu entsprechende Kante u' von a' nach dem erhaltene Senkenknoten eingefügt und am Ende wird die alte Kante u gelöscht. Der Graph wird am Ende so sehen:

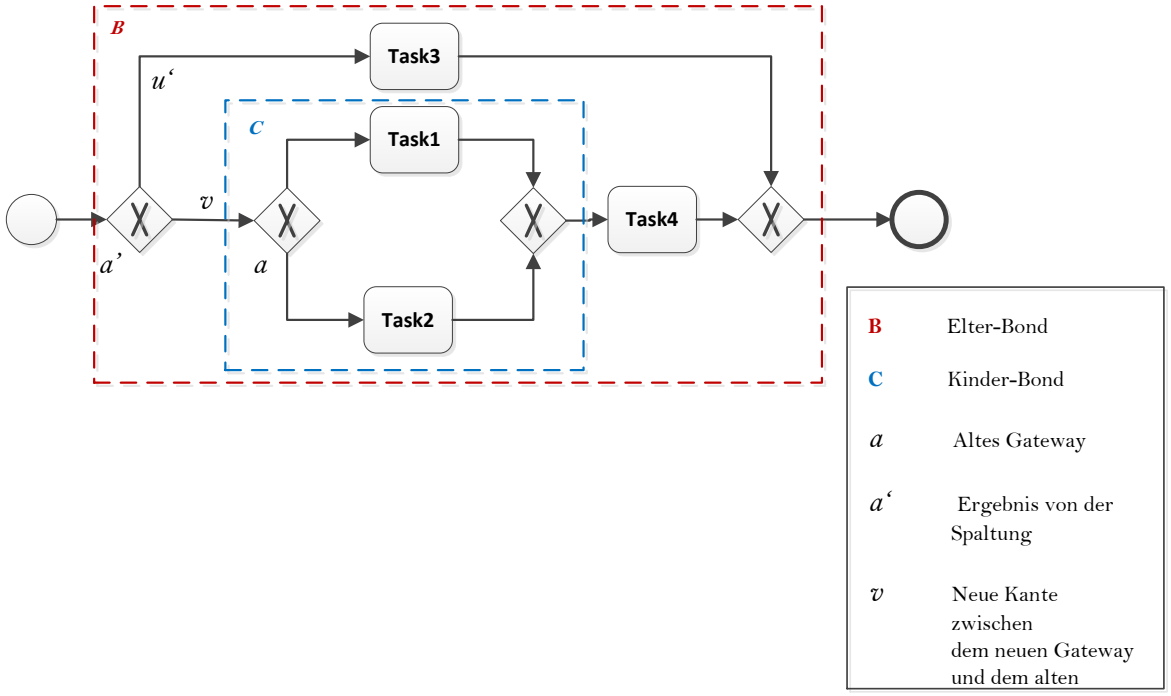


Abbildung 3.19: Selbes Beispiel von Abbildung 3.18 nach der Spaltung des Gateways und die Umleitung der einbezogenen Kanten

Dieser Ergebnisgraph ist semantisch gleich dem *Quasi structured* Eingangsgraph und ändert nicht den Fluss der Ausführung, so dass die Zweige in Gruppen aufgeteilt werden, abhängig von dem Gateway, in dem die Zweige enden. Und für jede Menge, sie mit einem entsprechenden Split-Gateway assoziieren. Wenn man eine *Quasi structured* Komponente analysiert, merkt man, dass diese Struktur nicht syntaktisch korrekt ist, da sie nicht äusgewogen‘ ist, d. h. es fehlt ihr ein „if“, der dem zweite „end if“ entsprechend ist. Und wenn man die Umstrukturierung macht, sieht es so aus dann:

if *Bedingung*3 **then**
    *task*3
else
    if *Bedingung*1 **then**
        *task*1
    else if *Bedingung*2 **then**

*task*2
end if
*task*4
end if

Anderseits, wenn der Elter und das Kind ein Gateway als Ausgangsknoten teilen, wie in Abbildung 3.20, wird das Gateway analog dem vorherigen Fall "gespaltet":

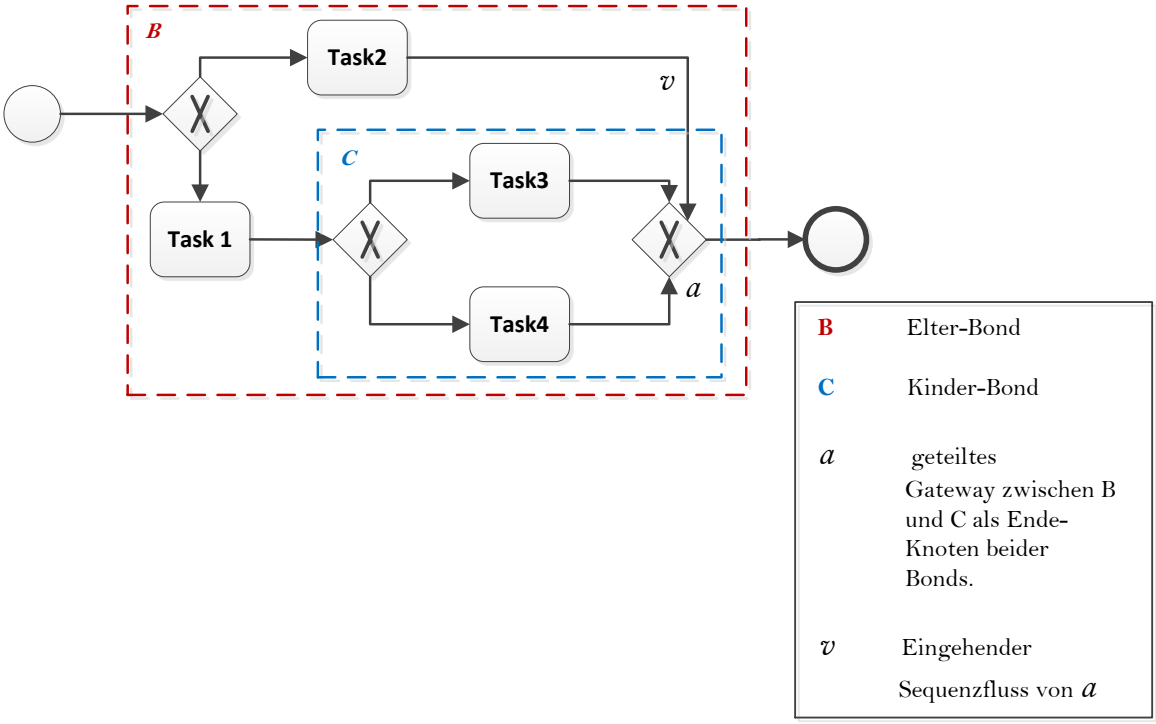


Abbildung 3.20: Beispiel von einer teilweise strukturierte Komponente wo Elter und Kind den Ausgangsknoten teilen

Ein neues Gateway a' mit demselben Typ vom geteilten Gateway wird erstellt, eine neue Kante vom alten Gateway a nach dem neuen Gateway a' wird eingefügt. Dann für jede eingehende Kante v vom a, die zum „Elter-Bond" B gehören und nicht zum Kind C, wird den Quellenknoten erhalten, eine neue entsprechende Kante v' vom erhaltenen Knoten nach a' eingefügt und am Ende wird die alte Kante v gelöscht.

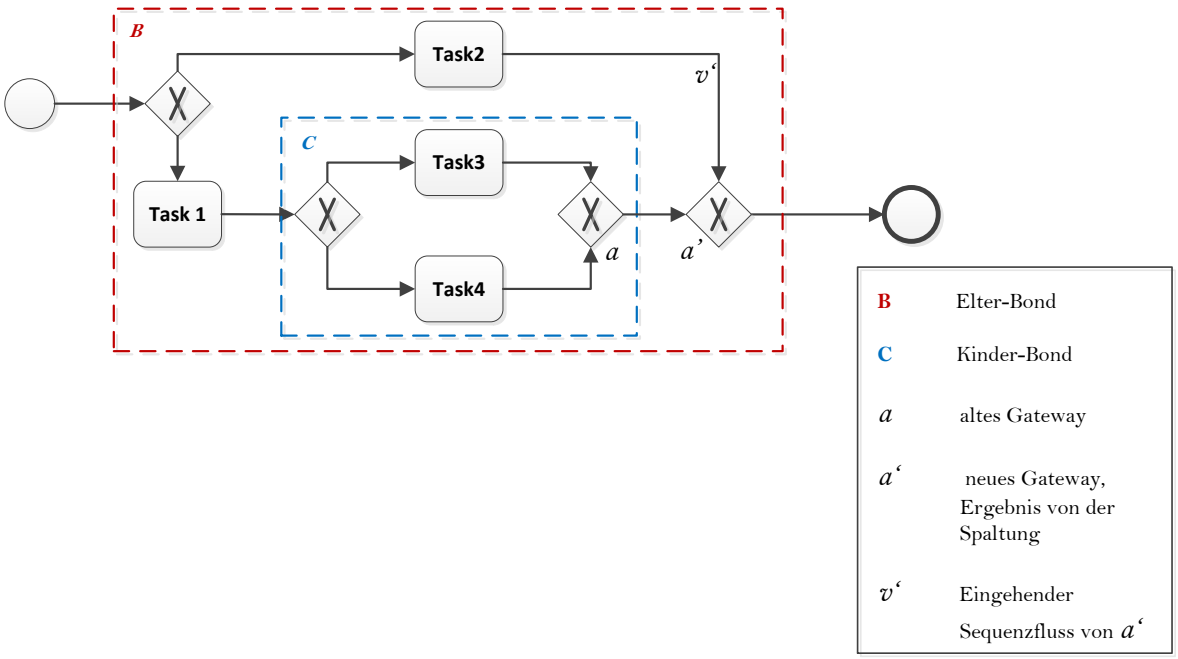


Abbildung 3.21: Selbes Beispiel von Abbildung 3.20 nach der vorherigen Transformation

Alle diese Veränderungen werden auf dem der entsprechenden Komponente Graph jedes Kindes durchgeführt und danach auf dem Workflow Graph. Wenn die Änderungen auf der Komponente durchgeführt werden, wird der entsprechende RPST nochmal abgerechnet, um die Umstrukturierungsfunktion rekursiv aufzurufen und so mehr Teilweise strukturierte Komponenten zu finden, falls es gibt. Sobald der ganze Graph umstrukturiert wird, wird der RPST aktualisiert, der dieses Mal *well structured* sein wird, damit die Überführung in BPEL gemacht werden kann.

3.4.2.2 Unstrukturierte

Diese Kategorie besteht aus alle Komponenten, die eine beliebige Anordnung haben und die normalerweise Gateways verbinden, die keine Umlenkungsfunktion übernehmen oder beide Funktionalitäten von Split und Join haben. Diese Strukturen können homogen (nur XOR Gateways oder AND Gateways haben) oder heterogen (eine Mischung zwischen XOR und AND Gateways) sein, in Abbildung 3.22 zeigen wir eine Klassifizierung laut Dumas [DGBP10] von allen möglichen Elementen eines RPSTs, mit dem Schwerpunkt in diesem Fall auf der Rigid Struktur.

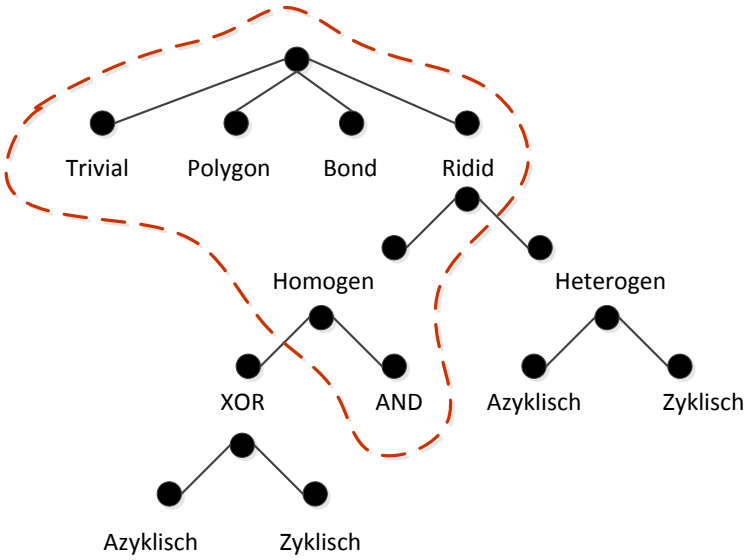


Abbildung 3.22: Klassifizierung von möglichen enthaltenen Elementen eines RPSTs und die Menge von betrachteten Elementen dieser Arbeit

Im Falle dieser Arbeit werden nur die homogenen unstrukturierten Strukturen berücksichtigt. Die Idee dieses Umstrukturierung-verfahrens ist die Struktur zu vereinfachen. Um diese Strukturen umzustrukturieren, werden zuerst die „Rigid“ Komponenten im RPST identifiziert. Danach als erster Schritt werden die Kinder von der Rigid Komponente analysiert. Alle triviale Komponenten, die zwei parallele Gateways verbinden, werden überprüft und alle die die Folgende Bedingungen erfüllen werden gelöscht:

- Das enthaltene Element des Quellenknotens ein divergierendes Gateway ist und das enthaltene Element des Senken-knotens ein konvergierendes Gateway ist.
- Das enthaltene Element des Quellenknotens ein divergierendes Gateway ist und das enthaltene Element des Senken-knotens ein gemischtes (konvergierend und divergierend) Gateway ist.
- Das enthaltene Element des Quellenknotens ein gemischtes Gateway ist und das enthaltene Element des Senken-knotens ein gemischtes Gateway ist.
- Das enthaltene Element des Quellenknotens ein gemischtes Gateway ist und das enthaltene Element des Senken-knotens ein konvergierendes Gateway ist. ...

Anderseits, alle trivialen Komponenten, die zwei divergierende Gateways u und v verbinden, wie in Abbildung 3.23 gezeigt, werden durch die folgende Methode transformiert:

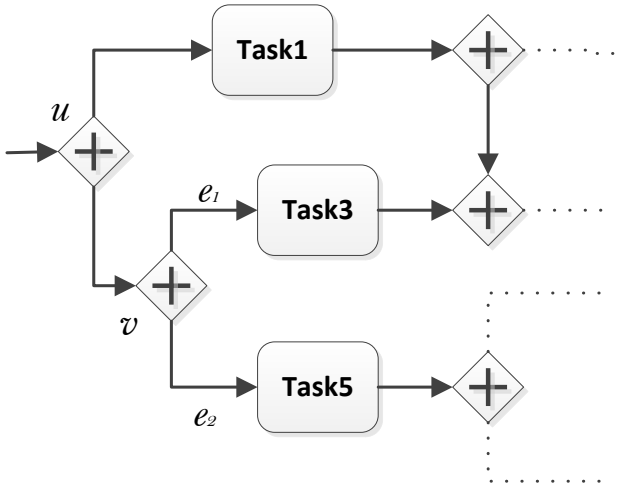


Abbildung 3.23: Ausschnitt vom im Kapitel 2.3 Generalized Flow Beispiel 2.12

Wenn so eine Komponente gefunden wird, werden alle ausgehende Kanten von v „umgelenkt“: Für jede ausgehende Kante e_i von v wird Senken-knoten s erhalten, in diesem Fall *Task3* und *Task5*, eine neue Kante e_i' von u nach s wird hinzugefügt und die Kante e_i gelöscht. Am Ende wird die Kante von u nach v und das verbleibende Gateway v gelöscht. Die resultierende Struktur würde aussehen, wie die Struktur in Abbildung 3.23.

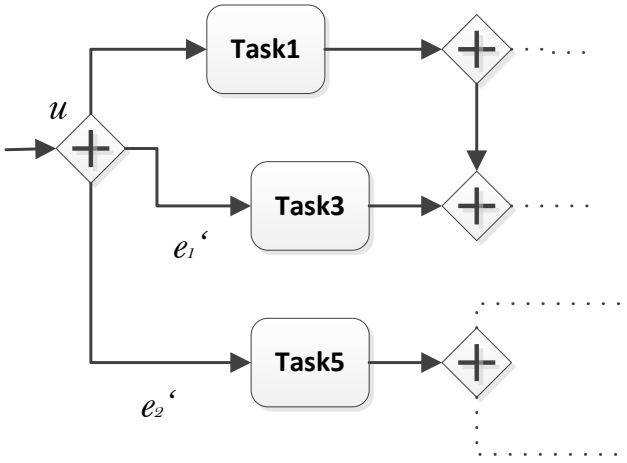


Abbildung 3.24: Ergebnis nach der Anwendung von den ersten Schritten der Transformation des 3.23 Generalized Flow Beispiels

Als letztes um die „Rigid“ Komponente zu lösen, werden die Kinder von der „Rigid“ erhalten, die „Polygon“ sind und die ein konvergierendes Gateway als Eingangsknoten haben. Die erste eingehende Kante vom Gateway wird nach dem Gateways folgenden Knoten umgelenkt (in diesem Fall, gibt es nur einen folgenden Knoten, da das Gateway konvergierend ist). Danach, werden die andere eingehende Kanten nach dem nächstgelegenen Gateway umgelenkt. Schließlich wird das verbleibende Gateway gelöscht, da es nicht mit dem Graph verbunden ist und keine Funktionalität hat.

Wenn alle diese Änderungen durchgeführt werden, wird der RPST des geänderten Fragments nochmal berechnet, um den nochmal zu traversieren und andere unstrukturierte Komponenten zu umstrukturieren, die nach den vorherigen Änderungen gebildet werden haben könnten.

4 Mapping Sets und State Propagation Rules

In diesem Kapitel wird sowohl die Begriff von *State Propagation Rules* erläutert als auch die Typen von *State Propagation Patterns* erklärt um später hinzuweisen welche *Patterns* von diesen im State-Propagation-Verfahren von BPEL2.0 nach BPMN2.0. Dann werden die möglichen Zustände von BPMN und BPEL vorgestellt und das Verfahren wird durch Schritte und Beispiele erklärt.

Wenn ein GeschäftsProzess in einer Sprache modelliert wird, braucht man öfter die Ausführung dieser Prozesse zu überwachen bezüglich des Zustands. Aber manchmal sind die Sprachen vom ausführenden Prozess und vom modellierten Prozess verschieden oder der Prozess ist in einer höheren Abstraktionsebene modelliert bevor eine verfeinerte Version des Prozesses zur Ausführung gebracht wird. Deshalb sollen die Überwachungsanwendungen sich darum kümmern, Prozesse unterstützen, die mit verschiedenen Sprachen spezifiziert sind, als die Sprache vom ausführenden Prozess, oder in verschiedenen Abstraktionsebenen modelliert wurden.

Dazu ist die *State Propagation* benutzt, um die Korrespondenz zwischen den Elementen einer Version des Prozesses und den Elementen einer anderen Version desselben Prozesses zusätzlich mit dem entsprechenden Zustand jedes hinzudeuten. Mit dieser Darstellung von den Korrespondenzen kann die Überwachung von einem Prozess in anderen Sprachen unterstützt werden.

Schumm et al. [SKLL11] stellen eine Klassifizierung von *State Propagation Patterns* vor, die die elementare Weise sind, in denen die Überführung von *Execution States* von Aktivitäten eines Prozesses-Instanz-Model in Aktivitäten eines anderen. Diese sind:

- Direct state propagation: Diese ist verwendbar wenn beide Prozesse sind auf derselben Sprache modelliert und haben dieselbe Zustand Menge.
- State Alteration: Diese wird benutzt wenn beide Prozesse in verschiedene Abstraktionsebenen modelliert sind, hier ändern die Typen von Zuständen wegen der Änderung von der Semantik der Aktivitäten oder von der Menge von Zustände ändern.
- State combination: Diese beschreibt eine Zusammenfügung von mehrere Zustände in einen Zustand, der danach mit einer Aktivität in einer höheren Abstraktionsebene assoziiert wird.
- State deduction: Diese repräsentiert der Fall in dem externe Daten sind benutzt, um den Zustand eines Element in einer höheren Abstraktionsebene zu deduzieren.
- Transition State: Diese beschreibt die Überführung eines Zustands in einen Kontroll-Link.

- State Aggregation: Diese beschreibt eine Überführung von mehreren Zuständen in einen im höheren Abstraktionsebenemodel.

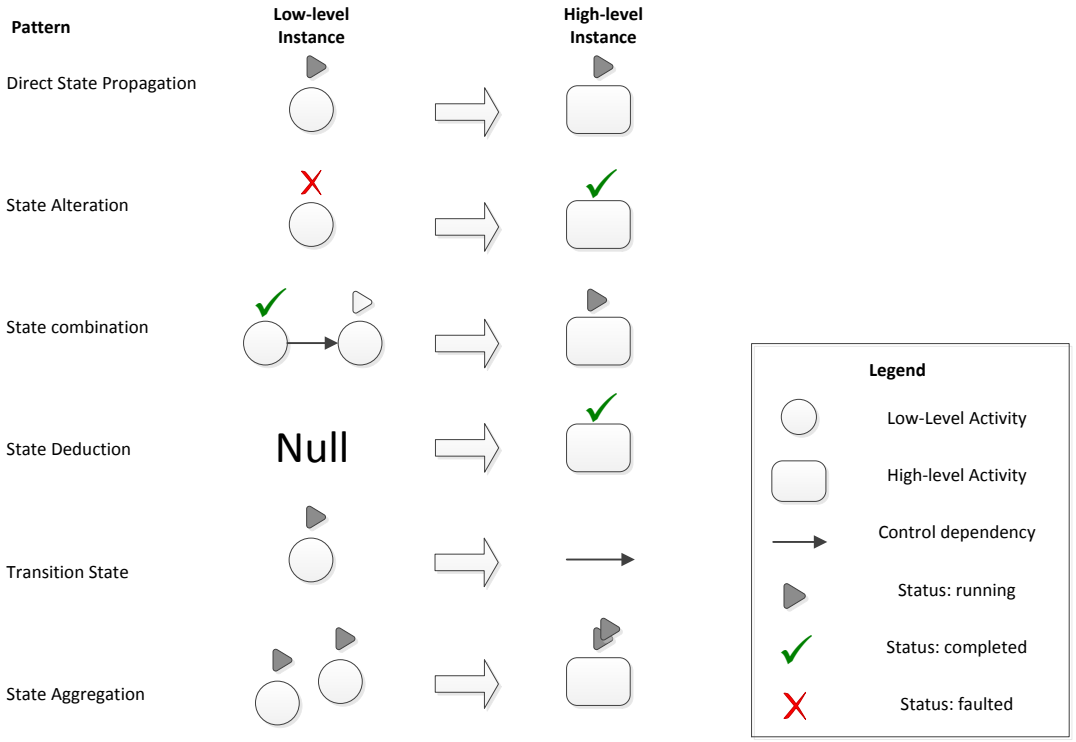


Abbildung 4.1: State Propagation Paterns definiert von Schumm et al. in [SKLL11]

In diesem Fall werden allgemein die „Direct State Propagation“ und die „State Alteration“ Regeln benutzt. Das heißt das für jede Aktivität, Subprozess mit dem zusätzlichen Zustand wird in eine Aktivität in den anderen Sprache überführt, der Zustand vom ersten Prozess-Modell kann dem Zustand des anderen direkt entsprechen oder diese kann ändern wegen der Änderung der Semantik.

Zuerst analysieren wir welche die möglichen Zustände der Elemente von einer BPMN-Prozess-Ausführung und einer BPEL-Prozess-Ausführung sind. Die Grundzustände von einer BPMN-Aktivität in einem bestimmten Zeitpunkt zur Vereinfachung werden in dieser Arbeit von 4 Zuständen ausgegangen sind: „laufend“, “abgeschlossen“, „wartend“, „fehlgeschlagen“ und „Nicht gestartet“. Als Startpunkt nehmen wir die Liste von Zuständen von BPEL-Aktivitäten die Schumm [KHK+11] vorstellt, und definieren unsere eigenen Zustände. Auch in dem Fall von BPEL vereinfachen wir die Zahl der Zustände auf vier und die sind: „aktiv“, „abgeschlossen“, „fehlgeschlagen“ und „inaktiv“. Wenn die Zustände von beiden Sprachen schon definiert sind, definiert man die *State Propagation Sets* die die Korrespondenz zwischen

den Zustände von einer Sprache und den Zustände von der anderen hindeuten. Hier werden 2 Sets definiert:

- Abwandlung 1:
  - Fehlgeschlagen → Fehlgeschlagen
  - wartend, laufend → aktiv
  - abgeschlossen → abgeschlossen
  - Nicht gestartet → inaktiv
- Combination-Teilprozess:
  - Alle enthaltene Elemente-Zustände sind „Abgeschlossen“ → Abgeschlossen
  - Zumindest ein der Zustände ist „inaktiv“ oder „wartend“ → Laufend
  - Zumindest ein der Zustände ist „fehlgeschlagen“ → Fehlgeschlagen

Um die *State Propagations Sets* darzustellen und später in den *State Propagation Rules* zu benutzen, verwendet man die Schema für *State Projection Definitions*, die in [SKLL11] präsentiert ist.

Im Folgenden, präsentiert man ein Beispiel von den *State Propagation Rules* die hindeuten, welche Aktivität oder Element vom BPEL (fromActivity) welchem in BPMN entspricht (toActivity) und der Zustand wird abhängend von der verwendeten *Set* in einen anderen überführt.

Listing 4.1 Definition von den *State Propagation Rules* zwischen den

```
<statePropagationRules>
	<statePropagationRule name="rule1">
		<fromActivities>
			<activity identifier="id" value="_5" />
		</fromActivities>
		<toActivity>
			<activity identifier="name" value="Service Task" />
		</toActivity>
		<stateSet pattern="Abwandlung1" />
	</statePropagationRule>
	<statePropagationRule name="rule2">
		<fromActivities>
			<activity identifier="id" value="_8" />
		</fromActivities>
		<toActivity>
			<activity identifier="name" value="Intermediate Event"
			    />
		</toActivity>
		<stateSet pattern="Abwandlung1" />
	</statePropagationRule>
	<statePropagationRule name="rule3">
		<fromActivities>
			<activity identifier="id" value="_10" />
		</fromActivities>
		<toActivity>
			<activity identifier="name" value="Task" />
		</toActivity>
		<stateSet pattern="Abwandlung1" />
	</statePropagationRule>
</statePropagationRules>
```

Für andere Strukturen wie Teilprozesse sind die *State Propagation Rules* anders, dieses Mal werden andere *Mapping Sets* definiert. Im Fall von Teilprozesse nehmen wir an diesem Punkt an, dass die enthaltene Strukturen nur Aktivitäten oder andere Teilprozesse sind und definieren wir es so: Wenn zumindest eine enthaltene Aktivität vom Scope Teilprozesses den Zustand *laufend* oder *inaktiv* hat, wird der Zustand von der Überführung in BPMN (Teilprozess) auch *laufend* sein. Auf der anderen Seite wenn der Zustand allen enthaltenen Elementen *abgeschlossen* ist, ist der Zustand der Überführung in BPMN (process) auch *abgeschlossen*. Das stellt man so mithilfe der Schema für *Projection Definitions* dar:

Listing 4.2 Darstellung von der Propagation Set „Combination-Teilprozess“

```
sss
1       <statePropagationSets>
2               <statePropagationSet name="Combination-TeilProzess">
3                       <condition>
4                               <containState>Wartend, Laufend</containState>
5                               <targetState>Aktiv</targetState>
6                       </condition>
7                       <condition>
8                               <allStatesEqual>Abgeschlossen</allStatesEqual>
9                               <targetState>Abgeschlossen</targetState>
10                      </condition>
11                      <condition>
12                              <containState>Fehlgeschlagen</allStatesEqual>
13                              <targetState>Fehlgeschlagen</targetState>
14                      </condition>
15                      <else></else>
16              </statePropagationSet>
17              ...
18        </statePropagationSets>
```

Um die Überführungsfällen von anderen Strukturen wie If, While oder Flow zu betrachten, braucht man eine Erweiterung von dem Schema der Propagation Sets zu machen, die diese unterstützten kann. Da diese BPEL-Strukturen in eine Sammlung von Aktivitäten zusammen mit Gateways überführt werden sollten. Und derzeitig ist das Schema so definiert, dass die eine Gruppe von Aktivitäten mit den entsprechenden Zuständen nach den Regeln in eine einzelne Aktivität überführt werden können, aber nicht andersrum.

Mit der ersten Alternative müssen einige Elemente umdefiniert werden, wie z. B. das <toActivity>, das eigentlich <toActivities> sein sollte, um ein BPEL-Element in mehreren BPMN-Elemente überführen zu können. In diesem Fall sollten auch die Bedingungen umdefiniert werden, damit in der entsprechenden *Propagation Set* beide genaue Zustandszuweisungen, vom Split- und Join-Gateway), stehen und danach diese „Set“ in einer *Propagation Rule* benutzen, anstatt doppelte. So werde ein If in zwei Gateways in BPMN überführt: das Split-Gateway und das Join-Gateway. In diesem Fall wird die „If“ separat von den Aktivitäten jedes Zweigs analysiert und der Zustand zugewiesen. Die Zustände können einfach so zugewiesen werden:

- Propagation Pattern-If:
  - Der Zustand der ganzen If ist „inaktiv“ → Der Zustand beide Gateways (Split und Join) wird „inaktiv“
  - Der Zustand der If ist „aktiv“ → Der Zustand des entsprechenden Split-Gateways ist „abgeschlossen“ und der Zustand des Join-Gateways ist noch „inaktiv“.

– Der Zustand der If ist „abgeschlossen“ → Der Zustand beide Gateways (Split und Join) wird „abgeschlossen“.

Eine zweite Alternative, die man auch betrachten könnte, ist die Bedingungen von den *Propagation Sets* umzudefinieren, damit diese prüfen welcher der Zustand des If-Konstrukts ist und eine *Set* in einer *Propagation Rule* zu benutzen, die der Zustand dem Split-Gateway zuweist und andere *Set* in anderer *Propagation Rule* benutzen, die der Zustand dem Join-Gateway zuweist. Das Problem mit diesem Ansatz ist, dass es zwei verschiedene *Rules* für dieselbe Quell-Aktivität (If) gibt.

Diese beide Alternativen gelten auch für andere BPEL-Strukturen die BPMN Gateways einbeziehen, wie Flow, pick, usw. Und im Fall von Flow und pick sind die vorgeschlagene *Propagations Sets* gleich wie mit dem If-Konstrukt.

So muss man, ähnlich mit anderen BPEL-Strukturen machen, um neue Set Propagation Sets zu definieren, die andere Elemente zurücküberführt.

5 Architektur und Implementierung

In diesem Kapitel werden die Werkzeuge und Libraries erklärt und beschreibt, die zum Zweck der Implementierung benutzt wurden.

Diese Library wird als Kerne von dem Web-basierte graphischen Business Process Editor Oryx. Die JBPT Library wurde benutzt um Graphen zu modellieren, besonders das Paket org.jbpt.graph.abs, davon die gehörende DirectedGraph, AbstractDirectedGraph und AbstractDirectedEdge Klassen verwendet werden, um die erforderliche Strukturen zu erstellen. Diese Strukturen helfen, den ProcessTree-Graph und seine einbezogene Elemente zu repräsentieren. Diese Strukturen sind:

- WFEdge (Workflow Kante)
- WFNode (Workflow Knoten)
- BPMNProcessTree (Workflow Baum)

In Abbildung 5.1 wird das Klassendiagramm von den verwendeten und erzeugten Klassen gezeigt. In diesem Diagramm die Klasse „AbstractMultiDirectHyperGraph“ gehört zu der JBPT library und ist eine Generalisation der „BPMNProcessTree“ Klasse.

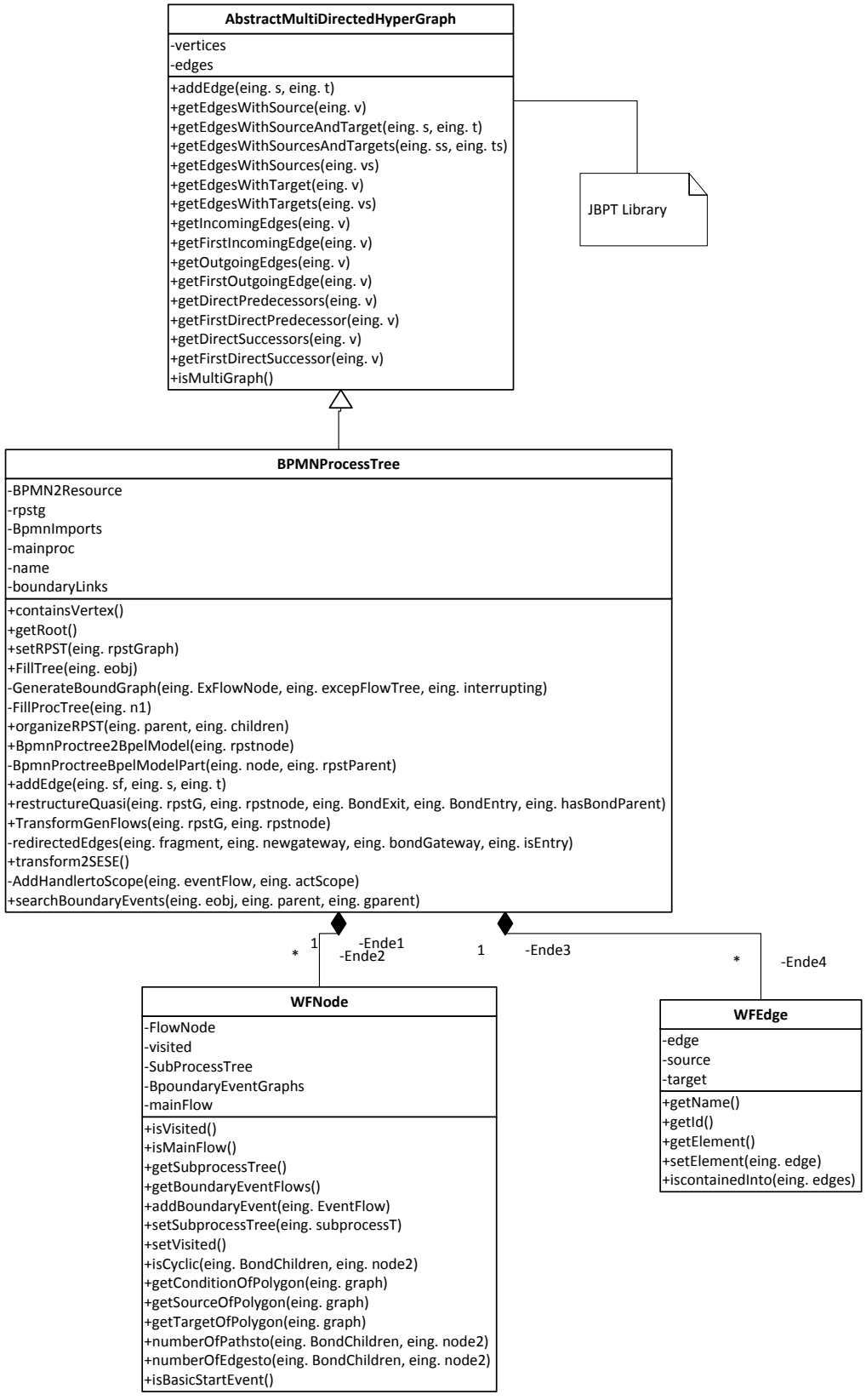


Abbildung 5.1: Klassendiagramm der beteiligten Klassen

Jede von diesen Objekten hat eine Struktur von Attributen und gehörenden Operationen. Die Operationen von der „BPMNProcessTree“ Klasse sind:

- containsVertex(): Prüft ob der Graph, einen bestimmten Knoten enthält.
- getRoot(): Holt den Start-Knoten (mit eingangsgrad „0“) des Graphs zurück.
- setRPST(): Verbindet ein RPST Graph mit dem BPMNProcessTree durch die Zuweisung dieses Werts dem rpst Attribut.
- FillTree(): Parst das XML-File entsprechend dem BPMN und macht die Elemente in einen Graph ein.
- GenerateBoundGraph(): Erzeugt einen BoundaryGraph,
- FillProcTree():
- organizeRPST(): organisiert die Kinder eines RPST nach der Reihenfolge, die diese Elemente graphisch haben.
- BpmnProcTree2BpelModel(): Ruft BpmnProcTree2BpelModelPart auf und gibt den BPEL-Prozess zurück, der entsprechend der BPMNTransformations ist.
- BpmnProcTree2BpelModelPart(): Analysiert den RPST und erzeugt die entsprechenden BPEL Konstrukte.
- addEdge(): Fügt eine neue Kante zu den Graph hinzu, die auf den entsprechenden Sequenzfluss verweist.
- restructureQuasi(): Sucht im Graph alle die *Quasi Structured Patterns* und strukturiert die um.
- TranforGenFlows(): Sucht im Graph alle die *Generalised Flows* und strukturiert die um.
- redirectEdges(): Wurde von restructureQuasi() aufgerufen um die eingehende oder ausgehende Kanten von den beteiligten Gateways umzuleiten.
- transform2SESE(): Transformiert einen Graph, der mehrere Start-Knoten bzw. Ende-Knoten hat, in einen SESE Graph.
- AddHandlertoScope(): Fügt einen neuen Handler zu einem bestimmten Scope.
- SearchBoundaryEvents(): sucht in einem Process-Element die BoundaryEvents und generiert die entsprechenden Handlers.

Die Operationen von der „WFNode“ sind:

- isVisited(): Prüft ob einen Knoten besucht wurde.
- isMainFlow(): Prüft ob einen Knoten zum Hauptflussgraph gehört.
- getSubProcessTree(): Gibt den gehörende Teilgraph zurück, falls der Knoten Instanz von „SubProcess“ ist.

- getBoundaryEventFlows(): Holt für diese BPMNProcessTree alle die BoundaryEvent-Graphen zurück.
- setSubProcessTree(): weist einen Teilgraph einem bestimmten Knoten.
- setVisited(): markiert einen Knoten als besucht.
- isCyclic(): Prüft ob eine Bond Komponente zyklisch ist.
- getConditionOfPolygon(): Für eine bestimmte Sequenz in einem Bond, erhält die Bedingung
- getSourceOfPolygon(): Holt den Quelleknoten der Sequenz zurück
- getTargetOfPolygon(): Holt den Zielknoten der Sequenz zurück.
- numberOfPathsto(): Erhält die Anzahl von Pfaden zwischen 2 Knoten.
- numberOfEdgesto(): Berechnet die Anzahl von Kanten zwischen 2 Knoten.
- isBasicStartEvent(): Prüft ob das Element dieses Knotens Instanz vom StartEreignis ist.

Und schließlich die Operationen von der „WFEdge“ sind:

- getId():

Um das BPMN File zu parsen und die da gefundene BPMN-Elemente zu analysieren, wird die Eclipse BPMN2 Library [BPM12] verwendet, die zum *Model Developement Tools* (MDT) gehört. Diese hilft das BPMN2 File in eine *Ressource* zu transformieren, die dieselbe hierarchische Struktur erhält und die DOM-Weise geparst sein kann. Diese vom File BPMN2 Elemente sind in einem Baum gespeichert, wo jedes Element in einem entsprechenden Knoten gekapselt wird.

Von derselben JBPT Library [JBP12] wird das org.jbpt.graph.algo.rpst Packet verwendet, um einen RPST von dem Graph zu bekommen. Dieses Packet enthält zwei wichtigen Elemente, die für diese Implementierung berücksichtigt werden: RPST und RPSTNode. Der RPST besteht aus eine Referenz auf das Wurzel-RPST-Knoten des RPSTs und enthält auch den gehörenden Graph und den Baum von Graphen, die zu einer bestimmten Komponente des RPSTs gehören. Der RPSTNode besteht aus einen Typ, der Startknoten und der Endeknoten des RPST-Knotens der auf Knoten vom BPMNProcessTree verweisen.

Um die analysierte BPMN-Elemente nach BPEL-Elemente zu transformieren verwendet man die BPEL Model Library, die hilft BPEL-Elemente zu repräsentieren und hierarchisch zu erzeugen. Wenn die Transformation schon durchgeführt wird, nimmt man das Wurzel BPEL-Element der alle andere geschachtelt enthält und mittels der BPEL-Writer schreibt den Prozess in einem File mit XML-Format.

Die mögliche Erweiterungen vom Code die gemacht werden können, um mehr BPMN-Konstrukte in BPEL überführen zu können, können in der *BpmnProctree2BpelModelPart* Funktion eingefügt werden. Im Fall der Aktivitäten, wenn man Unterstützung für Script-Aktivitäten, Benutzer-Aktivitäten oder manuelle Aktivitäten, kann man im Körper der

Verzweigung, die im Listing 5.1 vom Code der „BPMNProcessTree“ Klasse gezeigt wird, nach der Zeile 552 neue Bedingungen einfügen, die diese erwähnten Fälle betrachten.

Listing 5.1 Teil vom Code der BPMNProcessTree Klasse

```
547     if (node.getType().equals(TCType.T)) {
548             if (!node.getDescription().equals("Last-Trivial")) {
549                     // Call wftree2BpelModelPart
550                     WFNode nentry = (WFNode) node.getEntry();
551                     // If instance of normal Task
552                     if (nentry.getElement() instanceof Task) { ...
```

Im vorherigen Listing5.1 prüft man in der Zeile 552 ob die Struktur *Trivial* ist, in der Zeile 557 wird geprüft ob das enthaltene Element vom Knoten eine den Typ *Task* hat.

Im Fall von den unterbrechenden angehefteten Ereignisse die noch nicht unterstützt sind, fügt man den Code entsprechend der Erweiterung in der Methode *AddHandlertoScope* der *BPMNProcessTree* Klasse. Im Listing zeigen wir einen Ausschnitt vom Code der *AddHandlertoScope* Methode wo die Erweiterung bezüglich der angehefeteten Ereignisse eingefügt werden sollte.

Listing 5.2 Teil vom Code der AddHandlertoScope Methode

```
2546     else if (event.getEventDefinitions().get(0) instanceof
2547             EscalationEventDefinition) {
2548
2549                     // TODO
2550
2551   } else if (event.getEventDefinitions().get(0) instanceof
2552 SignalEventDefinition) {
2553
2554                     // TODO
2555       }
```

6 Zusammenfassung und Ausblick

Diese Arbeit wurde vom Bedarf eines Werkzeuges angetrieben, die eine Transformation von der Sprache BPMN2.0 nach BPEL durchführt. Um diese Transformation zu machen, musste man zuerst die verschiedene mögliche BPMN Strukturen analysieren und danach abhängig von der Struktur und den beteiligten Elementen, eine direkte Überführung von den Strukturen in BPEL anbieten oder die notwendige Verarbeitungen machen um diese danach zu transformieren.

In der Arbeit wurden Transformationen von drei Grundstrukturen gemacht. Diese Transformationen wurden mithilfe des RPST (Refined Process Structured Tree)erkannt. Diese drei Strukturen sind:

- *Well Structured Patterns*: Wo diese Struktur besteht aus allen Komponenten, die keinen Deadlock oder keine mehrfache Instanzen der Ausführung von einer selben Aktivität verursachen, und die direkt in BPEL ohne Bearbeitung überführt werden können.
- *Quasi Structured Patterns*: Sind alle die Komponenten, die aus nicht zusammengehörige Paare von Split- und Join-Gateways bestehen oder so zu sagen nicht ausgewogene Anzahl von Gateways haben. Diese sollten in *Well Structured Patterns* transformiert.
- *Generalised Flows*: Diese Strukturen sind azyklisch, haben keine bestimmte Strukturierung und verbinden normalerweise Gateways die keine Umlenkungsfunktion übernehmen.

In dieser Transformation wurde eine Aufteilung von dem *ProcessTree* in Komponenten mithilfe des RPST gemacht, um diese Komponenten auf der Suche nach den Grundpatterns später zu analysieren, die von Ouyang in citeODtHvdA06vorgestellt wurden. Die wichtigsten Beiträge dieser Arbeit, außer der Transformator von selbst, waren:

- Die Betrachtung in der Transformation von der Semantik der Aktivitäten und nicht nur von Struktur. z. B. eine wait Aktivität wird von einer Invoke Aktivität verschieden, die ist für dazu, dass mit den gezeigten Transformationen ausführbare BPEL Prozesse erzeugt werden können.
- Weiterhin wurde den Datenfluss betrachtet und in äquivalente BPEL Konstrukte überführt.

Die Konstrukte von BPMN die in dieser Arbeit, bezüglich der Transformation, unterstützt werden, sind folgende:

In der Kategorie der *Well structured patterns*: 1. Aktivitäten (Senden, Empfangend, None, Service) mit Schleife- und Parallele-Markern 2. Sequenzfluss (Besonderer Fall: Asoziationen) 3. Reihenfolge von Aktivitäten oder andere Konstrukte 4. Bond-Patterns: Starten und enden mit einer XOR-Gateway (einschließlich wohlgeformter Zyklen) Starten und enden mit einem Gateway-Inklusive Starten und enden mit einem Parallel-Gateway Starten mit einem Ereignisbasierten Gateway und enden in einem inklusiven Gateway 5. Teilprozesse 6. Eingetretene und Auslösende Ereignisse (Zeit, Nachricht, Fehler, Bedingung) 7. Unterbrechende angeheftete Zwischenereignisse (Boundaries) (Zeit, Fehler, Nachricht, Kompensation, Mehrfach) 8. Ereignis-Teilprozesse 9. EventHandlers . In der Kategorie der *quasi-structured patterns* sind es 1. Quasi mit XOR-Gateways 2. Quasi parallel mit Gateways . Diese werden in eine *well structured pattern* transformiert. Und schließlich wird in der Kategorie *Generalised flows* alles unterstützt, da sie nur aus parallelen Gateways bestehen.

Nach der Transformation wurden die *Mapping Sets* für behandelt, die von Schumm et al. [SKLL11] vorgestellt wurden. In dieser Arbeit wurden sie auf BPEL 2.0 und BPMN 2.0 erweitert. Diese geben durch die *State Propagation Rules* an, in welches Element (oder Elemente) eine Aktivität mit dem entsprechenden Zustand zugewiesen werden kann. Die Mapping Sets wurden für Aktivitäten, Scopes (von TeilProzessen) und Ifs betrachtet.

Die Elemente von BPMN, die die Transformation nicht unterstützt, sind: 1. User Aktivitäten, Skript Aktivitäten 2. Unterbrechende Ereignisse (Signal, Mehrfach parallel, Bedingung) 3. Nicht-unterbrechende angeheftete Ereignisse 4. Beliebige Zyklen .

Außer den Beiträgen, die in dieser Arbeit gemacht wurden, sind als Erweiterungspunkte für folgende Arbeiten empfohlen:

- Die Erweiterung der Transformator, damit die mehr BPMN-Strukturen unterstützt so wie:
  - Unterbrechende Boundary Events (Signal, Bedingung, Mehrfach Parallel)
  - Nicht-unterbrechende Boundary Events.
  - Beliebige Zyklen
  - Cross-Boundary Links
- Die Analyse für weitere Unterstützung der Mapping Sets von BPEL-Elementen nach BPMN-Elementen. Ebenso wie die Berücksichtigung von Neudefinitionen der Elemente der Schema.

Literaturverzeichnis

[BPM12] Eclipse BPMN2 Library, 2012. URL http://www.eclipse.org/modeling/mdt/?project=bpmn2. (Zitiert auf Seite 66)

[DGBP10] M. Dumas, L. García-Bañuelos, A. Polyvyanyy. Unraveling Unstructured Process Models. In J. Mendling, M. Weidlich, M. Weske, Herausgeber, *Business Process Modeling Notation - Second International Workshop, BPMN 2010, Potsdam, Germany, October 13-14, 2010. Proceedings*, Band 67 von *Lecture Notes in Business Information Processing*, S. 1–7. Springer, 2010. doi:http://dx.doi.org/10.1007/978-3-642-16298-5_1. (Zitiert auf den Seiten 23 und 53)

[Fou09] E. Foundation. BPEL to Java (B2J) Subproject, 2009. URL http://www.eclipse.org/stp/b2j/. (Zitiert auf Seite 9)

[FR10] J. Freund, B. Rücker. *Praxishandbuch BPMN 2.0*. Hanser, 2010. (Zitiert auf Seite 33)

[Gar08] L. Garcia Bañuelos. Pattern Identification and Classification in the Translation from BPMN to BPEL. In *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:*, OTM '08, S. 436–444. Springer-Verlag, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-88871-0_30. (Zitiert auf Seite 23)

[JBP12] JBPT Graph Library, 2012. URL http://code.google.com/p/jbpt/. (Zitiert auf Seite 66)

[KHB00] B. Kiepuszewski, A. H. M. ter Hofstede, C. Bussler. On Structured Workflow Modelling. In *CAiSE*, S. 431–445. 2000. (Zitiert auf Seite 16)

[KHK+11] O. Kopp, S. Henke, D. Karastoyanova, R. Khalaf, F. Leymann, M. Sonntag, T. Steinmetz, T. Unger, B. Wetzstein. An Event Model for WS-BPEL 2.0. Technical Report Computer Science 2011/07, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, University of Stuttgart, Institute of Architecture of Application Systems, 2011. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2011-07&engl=1. (Zitiert auf Seite 58)

[KMWL09] O. Kopp, D. Martin, D. Wutke, F. Leymann. The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. *Enterprise Modelling and Information Systems*, 4, 2009. (Zitiert auf Seite 9)

[Ley11] F. Leymann. BPEL vs. BPMN 2.0: Should You Care? In J. Mendling, M. Weidlich, M. Weske, Herausgeber, *Business Process Modeling Notation*, Band 67 von *Lecture Notes in Business Information Processing*, S. 8–13. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-16298-5_2. (Zitiert auf Seite 9)

[Ley12] P. F. Leymann. Vorlesung „Workflow 2“, 2012. (Zitiert auf den Seiten 7 und 29)

[MLZ06] J. Mendling, K. Lassen, U. Zdun. Transformation Strategies between Block-Oriented and Graph-Oriented Process Modelling Languages. In *Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006)*. GITO-Verlag Berlin, 2006. (Zitiert auf Seite 24)

[MR06] J. Mendling, J. Recker. On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. In *CAiSE 2006 Workshop Proceedings – Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2006)*. 2006. (Zitiert auf Seite 22)

[Obj11] Object Management Group (OMG). *Business Process Model and Notation (BPMN) Version 2.0*, 2011. URL http://www.omg.org/spec/BPMN/2.0/. OMG Document Number: formal/2011-01-03. (Zitiert auf Seite 11)

[ODHA06] C. Ouyang, M. Dumas, A. H. ter Hofstede, W. M. van der Aalst. Pattern-based Translation of BPMN Process Models to BPEL Web Services. *International Journal of Web Services Research (JWSR)*, 5(1):42–62, 2006. (Zitiert auf Seite 16)

[Sil11] B. Silver. *BPMN Method & Style With BPMN Implementer's guide*, Band 1. Cody-Cassidy Press, 2011. (Zitiert auf den Seiten 33 und 48)

[SKI09] S. Stein, S. Kühne, K. Ivanov. Business to IT Transformations Revisited. In D. Ardagna, M. Mecella, J. Yang, Herausgeber, *Business Process Management Workshops : BPM 2008 Internation Workshops, Milano, Italy, September 2008, Revised Papers*, Nummer 17 in LNBIB, S. 176–187. Springer, Berlin, 2009. (Zitiert auf Seite 23)

[SKLL11] D. Schumm, D. Karastoyanova, F. Leymann, S. Lie. Propagation of States from BPEL Process Instances to Chevron Models. Technical Report Computer Science 2011/06, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, University of Stuttgart, Institute of Architecture of Application Systems, 2011. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2011-06&engl=1. (Zitiert auf den Seiten 8, 57, 58, 59 und 70)

[VVK08] J. Vanhatalo, H. Völzer, J. Koehler. The Refined Process Structure Tree. In M. Dumas, M. Reichert, M.-C. Shan, Herausgeber, *BPM'08: Business Process Management, 6th International Conference, BPM 2008*, Band 5240 von *Lecture Notes in Computer Science*, S. 100–115. Springer, 2008. doi:10.1007/978-3-540-85758-7_10. (Zitiert auf den Seiten 13, 15, 19 und 47)

Alle URLs wurden zuletzt am 01.10.2012 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Omana Omar)