Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Diplomarbeit Nr. 3164

Transformation von BPEL Prozessmodellen in Grammatikbasierte Prozessmodelle

Karolina Vukojevic

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl. Inf. Katharina Görlach
begonnen am:	01.03.2011
beendet am:	31.08.2011
CR-Klassifikation:	F.4.2, H.4.1

Inhaltsverzeichnis

Abkürzungsverzeichnis

BPEL	Web Service Business Process Execution Language
EMF	Eclipse Modeling Framework
JAXB	Java Architecture for XML Binding
JRE	Java Runtime Environment
WoG	Workflow Grammatik Metamodell
WS	Web Service
WSDL	Web Service Description Language
XML	Extensible Markup Language

1 Einführung

Es existieren vielfältige Sprachen für die Beschreibung von Geschäftsprozessen als ausführbare Prozessmodelle. Diese Sprachen folgen unterschiedlichen Paradigmen, wie beispielsweise Datenfluss- oder Kontrollflussorientierung. Sie entstammen unterschiedlichen Anwendungsbereichen und fokussieren jeweils unterschiedliche Aspekte des zu beschreibenden Prozesses. Neben der Beschreibung von Geschäftsprozessen gewinnt in den letzten Jahren auch die Beschreibung von wissenschaftlichen Experimenten mit Hilfe von ausführbaren Prozessmodellen immer mehr an Wichtigkeit (Scientific Workflows) [TDG+07]. In diesem Kontext entstanden weitere Metamodelle für Prozessmodelle, die an diesen speziellen Anwendungsbereich angepasst sind.

Alle diese unterschiedlichen Sprachen beschreiben im Wesentlichen etwas sehr ähnliches, und zwar eine geordnete Abfolge von Aktivitäten. Um die unterschiedlichen Prozessbeschreibungssprachen zu klassifizieren und besser vergleichen zu können, wird am Institut für Architektur von Anwendungssystemen (IAAS) ein generisches Metamodell zur Beschreibung von Prozessen entwickelt. Ein Prozessmodell wird in diesem Metamodell als Grammatik dargestellt. Eine konkrete Prozessinstanz entspricht dann einem Wort der durch diese Grammatik beschriebenen Sprache. Das generische Metamodell wird im Folgenden als Workflow Grammatik Metamodell bezeichnet, ein Prozessmodell entspricht einer Workflow Grammatik.

Der Vorteil der Überführung von Prozessmodellen unterschiedlicher Metamodelle in Prozessmodelle des Workflow Grammatik Metamodells ist nicht nur die Vergleichbarkeit und einheitliche Klassifizierung von Prozessmodellen. Es ist ebenfalls möglich, die Ausführung von Prozessmodellen zu vereinheitlichen. Werden beliebige Prozessmodelle in Workflow Grammatiken transformiert, so wird lediglich ein Typ von Workflow Engine benötigt. Diese Workflow Engine muss in der Lage sein, Prozessmodelle des Workflow Grammatik Metamodells auszuführen.

In dieser Arbeit wird das Metamodell für ausführbare Workflow Grammatiken entworfen und ausspezifiziert. Es werden Konzepte und Konstrukte entwickelt, um Prozessmodelle als Workflow Grammatiken darzustellen. Zusätzlich wird ein Dateiformat für Workflow Grammatiken definiert.

Im zweiten Teil dieser Arbeit wird eine Transformation von BPEL Prozessmodellen in Workflow Grammatiken entwickelt. Es wird zunächst ein allgemeines Verfahren zur Abbildung von BPEL Prozessmodellen auf Workflow Grammatik Regelmengen vorgestellt. Anschließend wird für alle BPEL Aktivitätstypen beschrieben, wie sie auf Workflow Grammatiken abgebildet werden können. Im letzten Teil dieser Arbeit wird die Implementierung der Transformation von BPEL Prozessmodellen in Workflow Grammatiken vorgestellt.

1.1. Verwandte Arbeiten

Das Workflow Grammatik Metamodell soll als allgemeingültige Sprache zur Beschreibung von Prozessmodellen entworfen werden. In [AH10] wird ein Überblick über allgemeine Modellierungskonstrukte zur Darstellung von Prozessmodellen gegeben.

Workflow Grammatiken basieren auf dem Konzept allgemeiner formaler Grammatiken. Prozessmodelle können damit formal beschrieben werden. Eine weitere Möglichkeit, Prozessmodelle auf Basis eines formalen Modells darzustellen, sind Petri Netze. In [Aal98] wird die Verwendung von Petrinetzen zur Modellierung von Workflows beschrieben. In [Sta05] wird eine vollständige Abbildung von BPEL Prozessmodellen auf Petrinetze vorgestellt.

In Workflow Grammatiken wird der Kontrollfluss immer explizit modelliert. BPEL dagegen enthält in vielen Fällen impliziten Kontrollfluss. Bei der Transformation von BPEL Prozessmodellen in Workflow Grammatiken muss der implizite Kontrollfluss von BPEL in expliziten Kontrollfluss übersetzt werden. Eine ausführliche Analyse des Kontrollflusses in BPEL Prozessmodellen wird in [OVA+07] vorgenommen. In dieser Arbeit wird ebenfalls eine Abbildung von BPEL Prozessmodellen auf Petrinetze beschrieben.

1.2. Kapitelübersicht

In Kapitel 2 werden die für diese Arbeit wichtigen Grundlagen vorgestellt. Es wird eine Einführung in das Thema der Formalen Sprachen und Grammatiken gegeben. Der Begriff des Workflows wird eingeführt und damit verbundene Themen wie BPEL und Web Services beschrieben.

In Kapitel 3 wird das Metamodell für ausführbare Workflow Grammatiken entworfen. Am Anfang des Kapitels wird eine Motivation für den vorgestellten Entwurf gegeben. Zudem wird der Zusammenhang zwischen Grammatiken und Prozessmodellen veranschaulicht. In den folgenden Abschnitten werden die Aktivitätstypen von Workflow Grammatiken, die elementaren Modellierungskonstrukte sowie einige erweiterte Konstrukte spezifiziert und detailliert beschrieben.

In Kapitel 4 wird ein XML basiertes Dateiformat für Workflow Grammatiken definiert. Die konkreten Elementdefinitionen werden während der in Kapitel 5 beschriebenen Transformation von BPEL Prozessmodellen in Workflow Grammatiken vorgestellt.

In Kapitel 5 wird die Transformation von BPEL Prozessmodellen in Workflow Grammatiken entworfen. Zu Beginn wird ein allgemeines Vorgehensmodell entwickelt, mit dessen Hilfe einzelne BPEL Aktivitäten losgelöst vom restlichen Prozessmodell transformiert werden können. Anschließend wird für alle BPEL Aktivitätstypen eine Transformation in Workflow Grammatiken beschrieben und an Beispielen verdeutlicht

In Kapitel 6 wird die Implementierung der in Kapitel 5 beschriebenen Transformation vorgestellt. Es wird ein Überblick über den Aufbau der Implementierung gegeben. Die zentrale Methode der Transformation wird detailliert betrachtet.

In Kapitel 7 wird die Arbeit abschließend zusammengefasst. Die wesentlichen Aspekte werden nochmals hervorgehoben und es wird ein Ausblick auf mögliche weitere Arbeiten gegeben.

2 Grundlagen

2.1. Formale Sprachen

Die in diesem Kapitel verwendeten Definitionen orientieren sich an [Sch01]. Eine formale Sprache ist eine Menge von Wörtern. Ein Wort besteht aus einzelnen Zeichen eines Alphabets. Ein Wort besitzt immer eine endliche Größe, Sprachen dagegen können unendlich viele Wörter enthalten. Um Sprachen algorithmisch behandeln zu können, werden endliche Beschreibungsmöglichkeiten für Sprachen benötigt. Eine endliche Beschreibungsmöglichkeit für Sprachen sind Grammatiken. Eine Grammatik beschreibt, wie Wörter erzeugt werden können. Alle Wörter, die mit einer Grammatik erzeugt werden können, bilden eine Sprache.

G = (NT, T, P, S)

NT	= Menge der Nichtterminale, muss mindestens das Startsymbol S enthalten
T	= Menge der Terminale, das Alphabet
P	= Menge der Produktionsregeln
S	= Startsymbol, Element der Nichtterminalmenge

Definition 1: Grammatik

In Definition 1 werden formale Grammatiken definiert. Die Menge der Terminale bildet das Alphabet, jedes Wort besteht also ausschließlich aus Terminalen. Die in der Menge NT enthaltenen Nichtterminale werden während der Produktion eines Wortes als Platzhalter oder Variable verwendet. Die Produktionsregeln geben an, wie ein Wort produziert werden kann. Das Startsymbol ist das Nichtterminal, mit dem die Produktion jedes Wortes beginnt.

G = (NT, T, P, S)

NT = { S, A, B } T = { a, b } P = { S → A B, A → a, B →b }

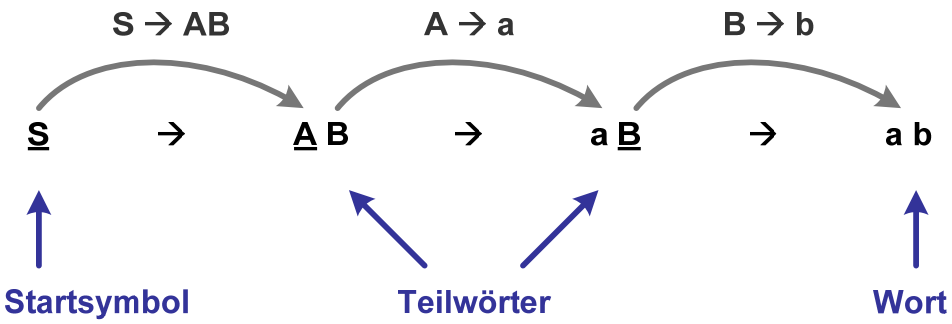


Abbildung 1: Beispiel einer Ableitung

Die Produktion eines Wortes bezeichnet man auch als Ableitung. Jede Ableitung beginnt mit dem Startsymbol. In jedem Ableitungsschritt wird eine Regel der Grammatik angewendet, dadurch entsteht ein neues Teilwort. Ein Teilwort besteht, im Gegensatz zum produzierten Wort, aus Nichtterminalen und Terminalen. Eine Regel besteht aus einer linken und einer rechten Seite und

beschreibt eine Ersetzung innerhalb eines Teilwortes. Die linke Seite der Regel wird durch die rechte Seite der Regel ersetzt.

Abbildung 1 beschreibt die Ableitung des Wortes „ab“ anhand einer Beispielgrammatik. Im ersten Schritt wird die Regel „S → AB“ angewendet. Das bedeutet, das Startsymbol wird durch die Nichtterminale „A“ und „B“ ersetzt. Auf das damit entstandene Teilwort „AB“ wird im nächsten Schritt die Regel „A → a“ angewendet. Das Nichtterminal „A“ wird durch das Terminal „a“ ersetzt, dadurch entsteht das Teilwort „aB“. Im letzten Schritt der Ableitung wird darauf die Regel „B → b“ angewendet. Damit ist die Produktion des Wortes „ab“ abgeschlossen.

2.1.1. Chomsky Hierarchie

Die Chomsky Hierarchie unterteilt Grammatiken in unterschiedliche Klassen bzw. Typen. Eine Grammatik beschreibt immer eine Sprache, dementsprechend klassifiziert die Chomsky Hierarchie nicht nur Grammatiken sondern auch Sprachen. Im Folgenden werden die einzelnen Klassen der Chomsky Hierarchie vorgestellt.

Jede Grammatik ist vom Typ 0. Die Menge der von Typ 0 Grammatiken erzeugten Sprachen ist die Menge der semientscheidbaren Sprachen.

Eine Grammatik ist vom Typ 1 oder kontextsensitiv, wenn für alle Produktionsregeln gilt, dass die rechte Seite nicht kürzer als die linke Seite ist. Die von Typ 1 Grammatiken erzeugten Sprachen werden kontextsensitive Sprachen genannt. Ein Beispiel für eine Regel einer kontextsensitiven Grammatik ist „aB → Ba“.

Eine Grammatik ist vom Typ 2 oder kontextfrei, wenn sie vom Typ 1 ist und zusätzlich für alle Produktionsregeln gilt, dass die linke Seite aus genau einem Nichtterminal besteht. Die von Typ 2 Grammatiken erzeugten Sprachen werden kontextfreie Sprachen genannt. Ein Beispiel für eine Regel einer kontextfreien Grammatik ist „B → BB“.

Eine Grammatik ist vom Typ 3 oder regulär, wenn sie vom Typ 2 ist und zusätzlich für alle Produktionsregeln gilt, dass die rechte Seite aus genau einem Terminal oder aus genau einem Terminal und einem Nichtterminal besteht. Die von Typ 3 Grammatiken erzeugten Sprachen werden reguläre Sprachen genannt. Ein Beispiel für eine Regel einer regulären Grammatik ist „B → bC“.

Die Chomsky- Hierarchie beschreibt eine echte Untermengenkonstruktion. Jede Typ 3 Grammatik ist beispielsweise immer auch eine Typ 2 Grammatik. Umgekehrt gibt es aber Typ 2 Grammatiken, die nicht vom Typ 3 sind. Die Menge der Typ 3 Grammatiken ist also eine echte Untermenge der Typ 2 Grammatiken.

Reguläre Grammatiken sind weniger mächtig als alle anderen Grammatiktypen. Viele Sprachen sind durch kontextsensitive oder kontextfreie Grammatiken beschreibbar, aber nicht durch reguläre Grammatiken. Die Chomsky Hierarchie unterteilt Grammatiken also auch nach ihrer Mächtigkeit. Die Typ 0 Grammatik ist am mächtigsten, die Typ 3 Grammatik am wenigsten mächtig.

Der Vorteil regulärer Grammatiken und Sprachen ist, dass sie algorithmisch gut handhabbar sind. Es ist beispielsweise entscheidbar, ob zwei reguläre Grammatiken die gleiche Sprache beschreiben. Für kontextfreie Grammatiken ist dieses Problem bereits nicht mehr entscheidbar. Andere Probleme sind für mehrere Klassen entscheidbar, dies aber mit jeweils unterschiedlicher Komplexität. Das Wortproblem („Ist ein Wort mit einer bestimmten Grammatik erzeugbar?“) ist für reguläre Grammatiken in linearer Zeit algorithmisch lösbar. Für kontextfreie Grammatiken ist es nur mit polynomiellem Aufwand algorithmisch lösbar.

Viele Programmiersprachen sind kontextsensitiv [Cla03]. Trotzdem wird ihre Syntax oft mit kontextfreien Grammatiken beschrieben. Die Aspekte der Programmiersprache, die damit nicht

beschreibbar sind, werden dann durch andere zusätzliche Mechanismen, wie beispielsweise Umgangssprache, erfasst.

2.2. XML

Extensible Markup Language (XML) ist eine Sprache zur Beschreibung von XML Dokumenten [XML08]. XML Dokumente werden zur strukturierten Darstellung von Daten verwendet. Die Strukturierung erfolgt hierarchisch mit Hilfe von Elementen und Attributen. XML Schema ist eine XML basierte Sprache zur Beschreibung von XML Dokumenten [XSD04].

2.3. Workflowtechnik

Geschäftsprozesse können durch Prozessmodelle beschrieben werden. Eine Instanz eines solchen Prozessmodells wird als Prozessinstanz oder Prozess bezeichnet. Diejenigen Teile eines Geschäftsprozesses, die durch einen Computer ausgeführt werden können, werden durch ein Workflowmodell beschrieben. Eine Instanz eines Workflowmodells wird als Workflow bezeichnet [LR00].

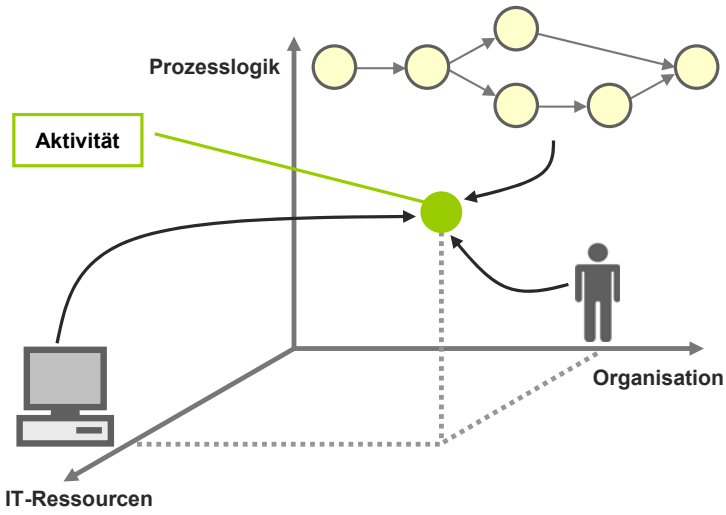


Abbildung 2: Drei Dimensionen eines Workflows

Wie in Abbildung 2 dargestellt, beinhaltet ein Workflowmodell drei Dimensionen, in dem eine Aktivität durch drei Achsen beschrieben wird. Die Dimension Prozesslogik beschreibt, wann welche Aktivitäten durchgeführt werden. Die Dimension IT-Ressourcen gibt an, womit eine Aktivität durchgeführt werden soll und die Dimension Organisation bestimmt, von wem die Aktivität durchgeführt werden soll [LR00].

2.4. Web Services

Web Services sind Anwendungen, die über Netzwerk Protokolle benutzbar sind [WS04]. Web Services basieren auf Web Standards, wie beispielsweise http, XML oder SOAP. Die Implementierung eines Web Services kann beliebig sein. Ein Web Service beschreibt nur die Schnittstelle, über die die Implementierung angesprochen werden kann [FZ09].

Web Service Description Language (WSDL) ist eine XML basierte Sprache zur Beschreibung von Web Services [WSDL07]. In WSDL werden sowohl die abstrakten Operationen als auch konkrete Formate und Protokolle zur Verwendung dieser Operationen definiert.

2.5. BPEL

Web Service Business Process Execution Language (WS-BPEL) ist eine XML basierte Sprache zur Beschreibung von Geschäftsprozessen [BPEL07]. BPEL Prozesse kommunizieren mit Webservices. Ein BPEL Prozess kann selbst wieder als Web Service zur Verfügung gestellt werden.

Zur Modellierung von BPEL Prozessen werden Aktivitäten verwendet. BPEL definiert zwei grundlegende Arten von Aktivitäten, Basis Aktivitäten und strukturierte Aktivitäten. Basis Aktivitäten sind atomare Aktivitäten. Strukturierte Aktivitäten enthalten andere Aktivitäten und erlauben so den Aufbau komplexer Prozessstrukturen [JMS06].

BPEL Prozesse besitzen eine Blockstruktur. Ein Prozess enthält genau eine Aktivität. Diese kann weitere Aktivitäten enthalten, die selbst wieder Aktivitäten enthalten können. Es ist in BPEL jedoch auch möglich, graphbasierte Strukturen zu modellieren. Dies ist nur innerhalb einer speziellen strukturierten Aktivität, dem „Flow“, möglich. BPEL kombiniert damit zwei unterschiedliche Modellierungsparadigmen.

In Kapitel 5 wird die Transformation von BPEL nach WoG beschrieben. An dieser Stelle werden auch die einzelnen BPEL Aktivitäten vorgestellt und beschrieben.

3 Workflow Grammatik Metamodell

Das Workflow Grammatik Metamodell (WoG) definiert, wie man mit einer Grammatik einen ausführbaren Prozess beschreiben kann. Eine WoG Grammatik entspricht damit einem Prozessmodel. Die Sprache, die durch eine WoG Grammatik beschrieben wird, ist die Menge aller möglichen Prozessdurchläufe. Ein Wort dieser Sprache stellt also eine konkrete Prozessausführung dar.

Der Entwurf von WoG verfolgt das Ziel, dass möglichst viele Workflowsprachen auf WoG abgebildet werden können. Aus diesem Grund wird WoG sehr feingranular aufgebaut. Jeder einzelne Schritt während der Ausführung eines Prozesses wird explizit modelliert. Damit ähnelt WoG in gewisser Weise einer Assemblersprache [Die05]. WoG besteht aus einer überschaubaren Menge von elementaren Aktivitäten und Konstrukten. Diese umfassen jedoch alle grundlegenden Funktionalitäten, die benötigt werden, um auch komplexere Strukturen modellieren zu können. Höhere Programmiersprachen, wie Java oder C++, werden durch einen Compiler ebenfalls auf einfache Sprachen mit einem reduzierten Befehlssatz, wie etwa Bytecode oder Assembler, übersetzt.

In der theoretischen Informatik wird eine Grammatik als ein Vier-Tupel, bestehend aus einer Menge von Nichtterminalen (NT), einer Menge von Terminalen (T), einer Menge von Produktionsregeln (P) und einem Startsymbol (S), definiert. Das Startsymbol S ist ein Element der Nichtterminalmenge NT [Sch01]. In dieser Arbeit gilt als Konvention, dass Nichtterminale immer mit Grossbuchstaben und Terminale immer mit Kleinbuchstaben dargestellt werden.

In WoG werden aktivierte Aktivitäten als Nichtterminale dargestellt, beendete Aktivitäten als Terminale. WoG unterscheidet dabei die Aktivitätstypen “Assign”, “Call”, „Evaluation“, „In“, „Out“, „Wait“ und „Quit“. Die Bezeichnung X_i beschreibt eine beliebige aktivierte Aktivität, ohne den Typen festzulegen. x_i beschreibt eine beliebige beendete Aktivität, ohne den Typen festzulegen. In Kapitel 3.1 wird die Bedeutung der unterschiedlichen WoG Aktivitäten näher erläutert. Neben den Nichtterminaltypen, welche die verschiedenen Aktivitätstypen repräsentieren, gibt es in WoG weitere Nichtterminaltypen. Diese werden in Kapitel 3.2 vorgestellt und näher beschrieben. Eine Übersicht über alle WoG Nichtterminaltypen findet sich in der Zusammenfassung in Kapitel 3.4.

Eine Regel einer WoG Grammatik beschreibt nicht nur eine Ersetzung innerhalb eines Teilwortes. Sie repräsentiert zusätzlich noch einen Schritt in der Ausführung eines Prozesses. Eine WoG Regel kann zum Beispiel beschreiben, dass eine Aktivität beendet wird oder dass eine Bedingung ausgewertet wird.

Die Anwendung einer Regel einer allgemeinen Grammatik erfolgt in einem Schritt, sie ist eine atomare Aktion. Die linke Seite einer Regel wird in einem Schritt durch die rechte Seite dieser Regel ersetzt. Daraus folgt für WoG, dass Aktionen, die bei der Prozessausführung in einer bestimmten zeitlichen Abfolge ausgeführt werden müssen, nicht in der gleichen Regel beschrieben werden dürfen. Der zeitliche Zusammenhang zwischen den einzelnen Aktionen geht sonst verloren. Wird zum Beispiel in einem Prozess die Aktivität X_1 beendet und der Kontrollfluss geht weiter zur Aktivität X_2, so werden in WoG dafür zwei Regeln definiert. Die eine Regel beendet die Aktivität X_1 und erzeugt einen Platzhalter P_{X2} für die Aktivität X_2. In der zweiten Regel wird nun mit Hilfe des Platzhalters P_{X2} die Aktivität X_2 erzeugt. Ein konkretes Beispiel wird in Kapitel 3.2.3 (Sequenz) beschrieben.

Die Ausführung von WoG Grammatiken durch eine Workflow Engine wird in [Hau11] betrachtet. Der Zusammenhang zwischen der Ausführung eines Prozesses und dem Erzeugen eines Wortes mit einer WoG Grammatik wird in Abbildung 3 anhand eines Beispiels noch einmal näher beschrieben. Von links nach rechts wird die schrittweise Ausführung eines Beispielprozesses

gezeigt. Oben in der Abbildung wird die Prozessausführung grafisch dargestellt. Aktivitäten werden durch Kreise dargestellt, Pfeile beschreiben den Kontrollfluss. Die noch nicht aktivierten Teile des Prozesses werden gestrichelt dargestellt. Unterhalb der grafischen Darstellung der einzelnen Prozessschritte wird die gleiche Prozessausführung in WoG dargestellt. Jedes Teilwort repräsentiert den gleichen Prozesszustand, wie er darüber grafisch dargestellt ist. Jeder Schritt der Prozessausführung wird durch eine Regel dargestellt. Welche Regel angewendet wird, um von einem in den nächsten Zustand zu kommen, wird in dieser Abbildung unterhalb der Teilwörter dargestellt.

Zu Beginn der Ausführung ist der gesamte Prozess inaktiv. In WoG wird dies durch das Startsymbol S dargestellt. Im ersten Schritt wird die Aktivität X_1 aktiviert. Das bedeutet in WoG, dass durch Anwendung von Regel (1) das Nichtterminal X_1 im Teilwort erzeugt und das Startsymbol gelöscht wird. Nach dem Ende der Aktivität X_1 wird der Kontrollfluss zu ihrer Nachfolgeaktivität X_2 aktiv. Dies wird in WoG durch Regel (2) beschrieben. Das Nichtterminal X_1 wird durch das Terminal x_1 ersetzt. Der ebenfalls erzeugt Platzhalter P_{X2} repräsentiert den Kontrollfluss zur nächsten Aktivität X_2. Die Aktivität X_2 wird anschließend aktiviert, in dem der Platzhalter P_{X2} durch X_2 ersetzt wird. Die restliche Prozessausführung erfolgt nach dem gleichen Prinzip.

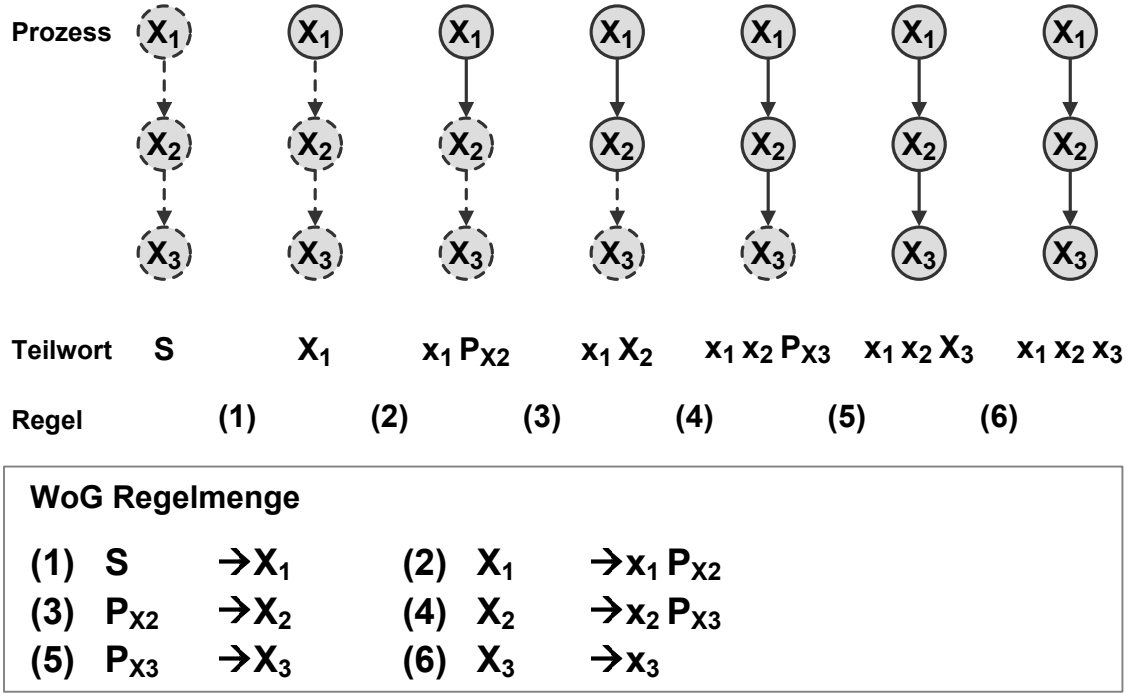


Abbildung 3: Beispiel einer Prozessausführung

Das Erzeugen eines Wortes einer WoG Grammatik entspricht einer sequentiellen Anwendung einzelner Regeln (auch Ableitung genannt). Eine Teilmenge dieser Regeln beschreibt das Aktivieren und Beenden einzelner Aktivitäten sowie die dazugehörenden Datenzugriffe. Betrachtet man diese Teilmenge von Regeln sowie die zeitliche Abfolge ihrer Anwendung, dann entspricht die Reihenfolge der Anwendung dieser Regeln einem gültigen Prozessdurchlauf. Die Produktionsregeln einer WoG Grammatik beschreiben die Struktur eines Prozesses. Ein von einer WoG Grammatik erzeugtes Wort beinhaltet alle während einer Prozessausführung aktivierten Aktivitäten. Aus dem Wort lässt sich weder die Prozessstruktur noch die genaue Aktivierungs- oder Beendigungsreihenfolge der Aktivitäten ablesen. Die Prozessstruktur wird durch die Produktionsregeln dargestellt, Details zur Ausführungsreihenfolge können über Logging Mechanismen protokolliert werden.

WoG beschreibt Kommunikation über Web Services. Das bedeutet, dass der Aufruf externer Programme immer ein Web Service Aufruf ist. Instanzen von WoG Prozessen werden wiederum als Web Services zur Verfügung gestellt. Als Datenmodell wird XML Schema verwendet. Ausdrücke werden mit XPath beschrieben. Die konkrete Ausgestaltung dieser Mechanismen wird in diesem Kapitel nicht betrachtet. Dies geschieht erst bei der Beschreibung der Transformation von BPEL nach WoG in Kapitel 5.

3.1. WoG Aktivitäten

WoG definiert verschiedene Arten von Aktivitäten. Diese werden im Folgenden vorgestellt und ihre jeweilige Funktion wird kurz beschrieben.

3.1.1. Call

Die Aktivität Call (C_i) ruft ein externes Programm auf. Bei dem Programmaufruf handelt es sich immer um einen Web Service Aufruf. Abbildung 4 stellt die Funktionsweise einer Call Aktivität schematisch dar. Der dargestellte Aufruf ist ein synchroner Aufruf. Um einen asynchronen Aufruf zu modellieren, benötigt man eine weitere Aktivität, die Aktivität „In“ (siehe Kapitel 3.1.2).

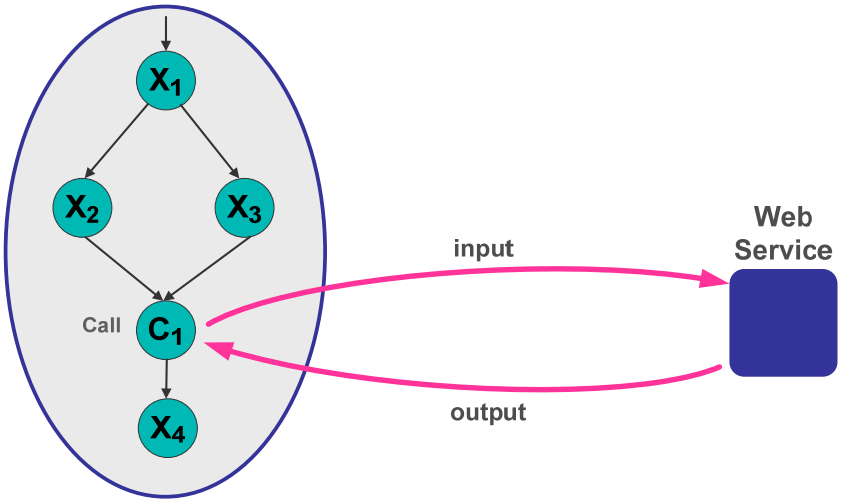


Abbildung 4: Aktivität „Call“

3.1.2. In

„In“ (I_i) ist eine Aktivität, die von außen aufgerufen werden kann und auf diese Weise Nachrichten empfängt. Über eine In Aktivität bietet ein Prozess nach außen hin seine Schnittstellen an. Prozessinstanzen werden durch den Aufruf einer In Aktivität erzeugt. Daher gibt es in jedem Prozess mindestens eine In Aktivität.

In Aktivitäten können beispielsweise auch benutzt werden, um einen asynchronen Aufruf zu modellieren. Abbildung 5 zeigt einen solchen asynchronen Aufruf. Die Aktivität C_1 ruft einen Web Service auf, wartet aber nicht auf das Ergebnis. Später im Prozess wird die Aktivität I_1 aktiviert. Diese wartet auf das Ergebnis des von C_1 aufgerufenen Web Services.

Eine In Aktivität kann nicht nur eine, sondern mehrere Schnittstellen definieren. Trotzdem empfängt ein In immer genau eine Nachricht. Sobald die erste Nachricht empfangen wird, werden alle Schnittstellen der entsprechenden In Aktivität deaktiviert.

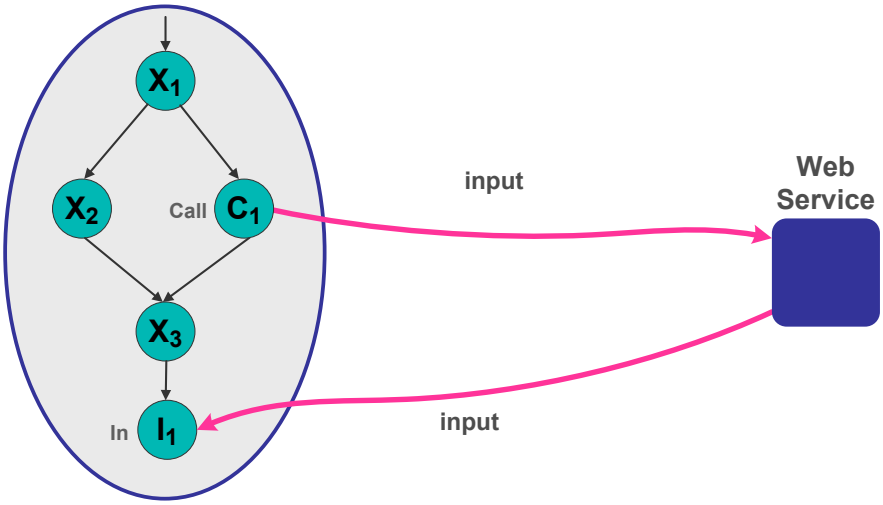


Abbildung 5: Aktivität „In“

3.1.3. Out

Ein Prozess bietet nach außen Schnittstellen an und kann über diese Nachrichten empfangen. Das Empfangen einer Nachricht wird, wie im vorigen Abschnitt beschrieben, über eine In Aktivität modelliert. Eine Schnittstelle definiert im Allgemeinen nicht nur Eingabe- sondern auch Ausgabeparameter. Man benötigt also eine Möglichkeit, nach dem Empfangen einer Nachricht eine dazugehörende Antwortnachricht zurückzusenden. Die Aktivität Out (O_i) modelliert genau dieses. Eine Out Aktivität gehört immer zu einer In Aktivität.

Abbildung 6 zeigt ein Beispielprozess, der von außen aufgerufen wird. Die Aktivität I_1 bietet eine Schnittstelle nach außen hin an, über die das aufrufende Programm den Prozess aufruft. Das aufrufende Programm wartet anschließend auf das Ergebnis, es handelt sich um einen synchronen Aufruf. Die Aktivität X_1 bearbeitet die über I_1 empfangenen Daten und erzeugt ein Ergebnis. Über die Aktivität O_1 wird dieses an das aufrufende Programm zurückgegeben.

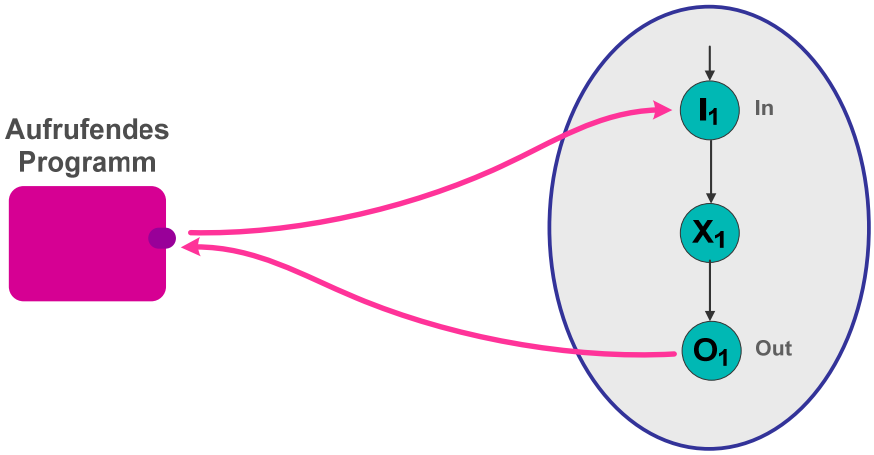


Abbildung 6: Aktivität „Out“

3.1.4. Assign

Mit der Aktivität „Assign“ (A_i) modelliert man in WoG eine oder mehrere Zuweisungen. Eine Assign Aktivität bietet einen transaktionalen Rahmen für die in ihr enthaltenen Zuweisungen. Dies wird in

Kapitel 5.4.4 näher beschrieben. Eine Zuweisung hat immer schreibenden Zugriff auf eine Variable und kann lesenden Zugriff auf eine oder mehrere Variablen haben.

3.1.5. Evaluation

Die Aktivität „Evaluation“ (E_i) wertet einen booleschen Ausdruck aus. Das Ergebnis des Ausdrucks ist immer „wahr“ oder „falsch“. In Kapitel 3.2.9 wird beschrieben, wie dieses Ergebnis benutzt wird, um bedingten Kontrollfluss zu modellieren.

3.1.6. Wait

Mit der Aktivität „Wait“ (W_i) kann „warten“ modelliert werden. Wenn eine Aktivität Wait aktiviert wird, bleibt sie eine vorgegebene Zeit aktiv, bevor sie beendet wird. Man kann zum einen die Zeitdauer definieren, wie lange die Aktivität Wait aktiv ist. Alternativ kann auch ein Zeitpunkt definiert werden. Das Wait ist in diesem Fall genau bis zu diesem Zeitpunkt aktiv.

3.1.7. Quit

Eine Aktivität vom Typ Quit (Q_i) repräsentiert das vorzeitige und explizite Beenden eines Prozesses.

3.2. Grundlegende WoG Konstrukte

Im Folgenden wird gezeigt, wie Prozesse mit Hilfe von WoG beschrieben werden können. Die betrachteten Teile eines Prozesses werden zur besseren Verständlichkeit grafisch dargestellt. Die Kreise stellen Aktivitäten dar. Die Pfeile beschreiben den Kontrollfluss zwischen den Aktivitäten. Rechtecke, welche auf Pfeilen platziert sind, repräsentieren Bedingungen, die den durch die Pfeile beschriebenen Kontrollfluss beeinflussen. Kreise, die den Prozess oder Teile des Prozesses umranden, symbolisieren den Lebensbereich der darin enthaltenen Variablen. Die eingeklammerten Nummern verweisen auf die Regeln der entsprechenden Regelmenge, die diesen Prozess in WoG modelliert.

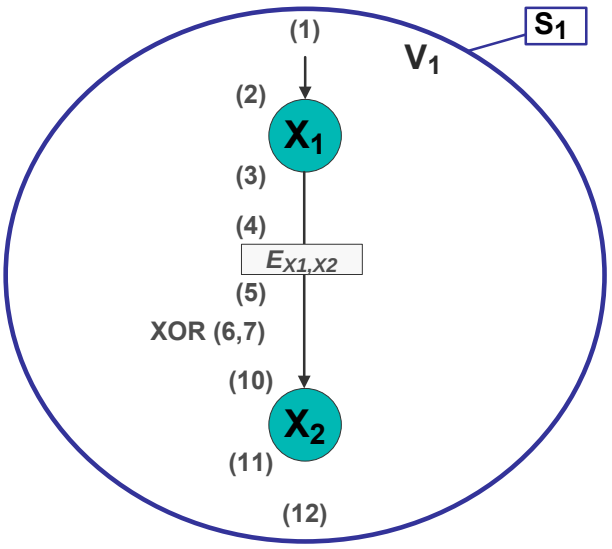


Abbildung 7: Darstellung eines Beispielprozesses

Abbildung 7 zeigt einen kleinen Beispielprozess, der die Aktivitäten X_1 und X_2 enthält. Der Kontrollfluss beginnt bei Aktivität X_1 und geht dann weiter zu X_2. Der Kontrollfluss zwischen X_1 und X_2 ist abhängig von der Bedingung $E_{X1,X2}$. Der gesamte Prozess ist von dem Bereich S_1 umgeben. Die Variable V_1 lebt innerhalb dieses Bereiches.

3.2.1. Aktivierung einer Aktivität

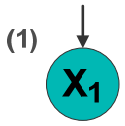


Abbildung 8: Aktivierung einer Aktivität

In Abbildung 8 wird die Aktivierung einer Aktivität X_1 dargestellt. In WoG wird dies mit folgender Regel beschrieben:

(1) … → … X_1 …

Regelmenge 1: Aktivierung einer Aktivität

Das Nichtterminal X_1 kommt auf der rechten Seite der Regel vor, auf der linken Seite dagegen nicht. Dies bedeutet, dass bei Anwendung der Regel das Nichtterminal X_1 im Wort neu erzeugt und die entsprechende Aktivität aktiviert wird.

3.2.2. Beendigung einer Aktivität

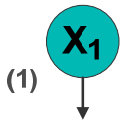


Abbildung 9: Beendigung einer Aktivität

Abbildung 9 stellt die Beendigung einer Aktivität X_1 dar. Die entsprechende Regel in WoG wird in Regelmenge 2 gezeigt.

(1) … X_1 … → … x_1 …

Regelmenge 2: Beendigung einer Aktivität

Das Nichtterminal X_1 kommt auf der linken Seite der Regel vor. Auf der rechten Seite steht an Stelle des Nichtterminals X_1 das Terminal x_1. Bei Anwendung der Regel wird das Nichtterminal X_1 aus dem Wort entfernt und das Terminal x_1 eingefügt sowie die entsprechende Aktivität beendet.

Ein Terminal innerhalb eines Wortes repräsentiert in WoG eine beendete Aktivität einer Prozessinstanz. Dies bedeutet, dass es in WoG keine Regeln gibt, in denen Terminale entfernt werden. Zusätzlich gilt, dass Terminale ihre Position im Wort in Bezug auf (aktivierte oder beendete) Aktivitäten nicht verändern. Dies ist eine Voraussetzung dafür, dass zwei gleiche Prozessdurchläufe auch das gleiche Wort erzeugen.

3.2.3. Sequenz

Eine Sequenz beschreibt eine Hintereinanderausführung von Aktivitäten. Die in Abbildung 10 dargestellte Sequenz sagt beispielsweise aus, dass die Aktivität X_2 erst dann aktiviert werden darf, wenn X_1 beendet worden ist. Dies entspricht einer sogenannten „Ende-Anfang-Beziehung“ [Ros06]. Solche Beziehungen werden in WoG durch Regeln realisiert, die mit Platzhaltern verknüpft sind.

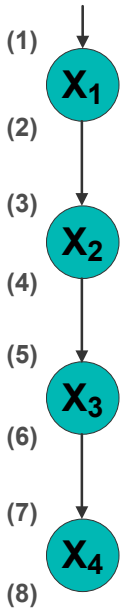


Abbildung 10: Sequenz von Aktivitäten X_1, X_2, X_3, X_4

In Abbildung 10 wird eine Sequenz von Aktivitäten X_1, X_2, X_3 und X_4 dargestellt. WoG beschreibt die Hintereinanderausführung dieser Aktivitäten mit folgenden Regeln:

(1)	S	→	X_1
(2)	X_1	→	$x_1 P_{X2}$
(3)	P_{X2}	→	X_2
(4)	X_2	→	$x_2 P_{X3}$
(5)	P_{X3}	→	X_3
(6)	X_3	→	$x_3 P_{X4}$
(7)	P_{X4}	→	X_4
(8)	X_4	→	x_4

Regelmenge 3: Sequenz von Aktivitäten X_1, X_2, X_3 und X_4

Das Startsymbol S aktiviert in Regel (1) die Aktivität X_1. In Regel (2) wird die Aktivität X_1 beendet und der Platzhalter P_{X2} für die Folgeaktivität X_2 erzeugt. Dadurch wird sichergestellt, dass die Aktivität X_2 erst dann aktiviert werden kann, wenn X_1 beendet worden ist. Der für die Aktivierung von X_2 notwendige Platzhalter P_{X2} wird nur dann erzeugt, wenn die Aktivität X_1 beendet wird. In Regel (3) wird mit Hilfe des Platzhalters P_{X2} die Aktivität X_2 aktiviert. Regel (4) beendet die Aktivität X_2 und erzeugt den Platzhalter P_{X3} für die Folgeaktivität X_3. Der Platzhalter P_{X3} aktiviert in Regel (5) die Aktivität X_3. In Regel (6) wird die Aktivität X_3 beendet und der Platzhalter P_{X4} für die

Folgeaktivität X_4 erzeugt. Mit Hilfe des Platzhalters P_{X4} wird in Regel (7) die Aktivität X_4 aktiviert. In Regel (8) wird die Aktivität X_4 beendet.

Die Anwendung der Regeln aus der Regelmenge 3 ist in Ableitung 1 dargestellt. Die Nummer über den Pfeilen zeigt, welche Regel in dem jeweiligen Ableitungsschritt verwendet wird. Befindet sich neben dieser Nummer ein Stern, so bedeutet dies, dass die Regel mehrmals angewendet wird. Nichtterminale, auf die eine Regel angewendet wird, sind hier zum besseren Verständnis unterstrichen dargestellt.

$$S \xrightarrow{(1)} \underline{X_1} \xrightarrow{(2)} x_1\,\underline{P_{X2}} \xrightarrow{(3)} x_1\,\underline{X_2} \xrightarrow{(4)} x_1\,x_2\,\underline{P_{X3}} \xrightarrow{(5)} x_1\,x_2\,\underline{X_3} \xrightarrow{(6)} x_1\,x_2\,x_3\,\underline{P_{X4}} \xrightarrow{(7)} x_1\,x_2\,x_3\,\underline{X_4} \xrightarrow{(8)} x_1\,x_2\,x_3\,x_4$$

Ableitung 1: Sequenz

Bei der Modellierung einer Sequenz sind die Platzhalter sehr wichtig. Die in Abbildung 11 dargestellten Regelmengen modellieren nicht das Gleiche. Die linke Regelmenge beschreibt in Regel (1) zwei Dinge gleichzeitig. Sie sagt aus, dass Aktivität X_1 beendet und Aktivität X_2 aktiviert wird. Sie sagt aber nicht aus, dass dies in einer bestimmten zeitlichen Reihenfolge geschieht. Eine Sequenz beschreibt aber eine klare zeitliche Abfolge von Aktivieren und Beenden von Aktivitäten. Dies wird durch die rechte Regelmenge mit Hilfe des Platzhalters P_{X2} korrekt modelliert.

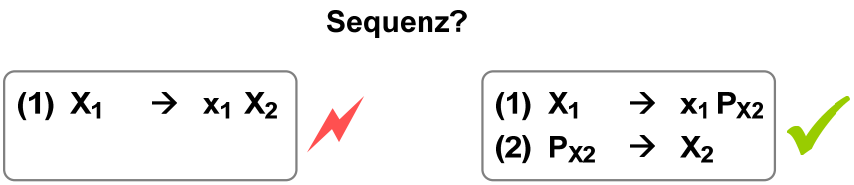


Abbildung 11: Verwendung von Platzhaltern

Die bisher gezeigten Konstrukte beschreiben den Kontrollfluss. In einem Prozess gibt es neben dem Kontrollfluss aber auch den Datenfluss. Wie der Datenfluss in WoG modelliert wird, ist im nächsten Abschnitt erläutert.

3.2.4. Variablen

In WoG werden Daten mit Hilfe von Variablen dargestellt. Aktivitäten greifen auf Variablen lesend oder schreibend zu.

Abbildung 12 stellt eine Aktivität X_2 dar, welche auf die Variablen V_1 und V_2 lesend und auf die Variable V_1 schreibend zugreift. Die beiden Rechtecke links von der Aktivität stellen diese Zugriffe bildlich dar. Rechts neben der Aktivität ist ihre Funktionalität näher beschrieben.

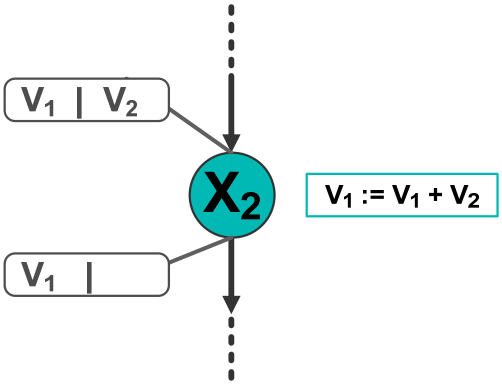


Abbildung 12: Input- und Outputvariablen einer Aktivität

WoG stellt alle Variablenzugriffe einer Aktivität in den Aktivierungs- und Beendigungsregeln dieser Aktivität dar. Die lesenden Zugriffe werden in der Aktivierungsregel und die schreibenden Zugriffe in der Beendigungsregel beschrieben. Durch diese Trennung ist es möglich, alle Kombinationen von Variablenzugriffen einer Aktivität darzustellen (nur lesend, nur schreibend, lesend und schreibend).

(1) $\dots V_1\ V_2\ P_{X2} \dots \rightarrow \dots V_1\ V_2\ X_2 \dots$

(2) $\dots V_1\ X_2 \dots \rightarrow \dots V_1\ x_2 \dots$

(I) $V_i\ Y \rightarrow Y\ V_i\ , \quad Y \in (NT \cup T)$

(II) $Y\ V_i \rightarrow V_i\ Y\ , \quad Y \in (NT \cup T)$

Regelmenge 4: Input- und Outputvariablen einer Aktivität

Regelmenge 4 stellt die lesenden und schreibenden Zugriffe der Aktivität X_2 dar. Damit die Aktivität X_2 aktiviert werden kann, benötigt sie lesenden Zugriff auf die Variablen V_1 und V_2. Dies wird mit Regel (1) modelliert. Regel (2) sagt aus, dass die Aktivität X_2 schreibenden Zugriff auf V_1 benötigt, um beendet werden zu können. Variablen tauchen hier immer auf beiden Seiten der Regeln auf. Ein Variablenzugriff erzeugt keine Variablen und löscht keine Variablen. Vielmehr existieren die Variablen bereits vor dem Zugriff und bestehen auch danach weiter. Das Erzeugen und Löschen einer Variablen wird in Kapitel 3.2.5 Scope eingeführt. Die Position der Variablen innerhalb der Regeln spielt keine Rolle. Es wird aber als Konvention gewählt, dass die Variablen immer links von der Aktivität (bzw. ihrem Platzhalter) stehen und sortiert sind.

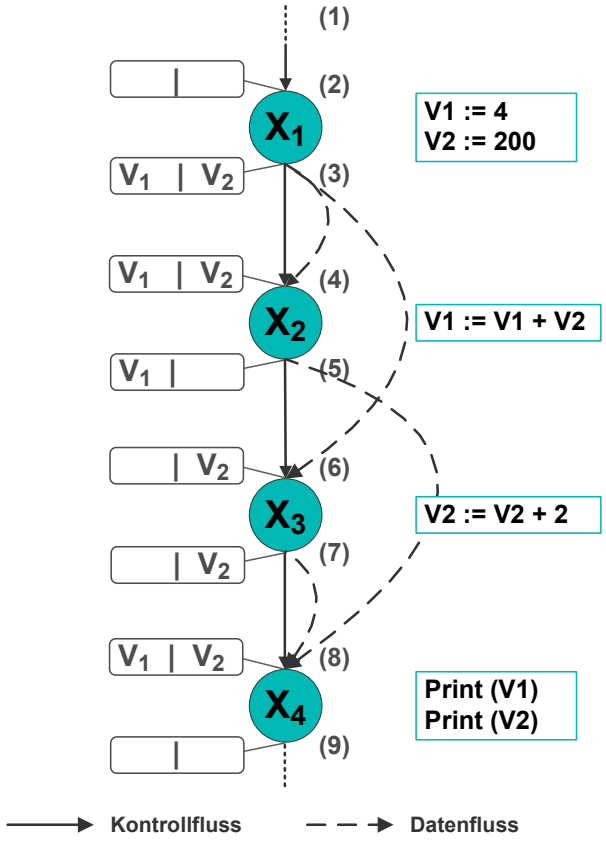


Abbildung 13: Datenfluss

Abbildung 13 zeigt den Datenfluss beispielhaft an einem Prozessausschnitt. Zur besseren Verständlichkeit wird der Datenfluss hier explizit mit einer gestrichelten Linie dargestellt. Die Aktivitäten X_1, X_2, X_3 und X_4 arbeiten mit den Variablen V_1 und V_2. Die Regelmenge 5 modelliert diesen Prozessausschnitt in WoG.

(1)	...	→	P_{X1}	Prozess-strukturregeln
(2)	P_{X1}	→	X_1	
(3)	$V_1\ V_2\ X_1$	→	$V_1\ V_2\ x_1\ P_{X2}$	
(4)	$V_1\ V_2\ P_{X2}$	→	$V_1\ V_2\ X_2$	
(5)	$V_1\ X_2$	→	$V_1\ x_2\ P_{X3}$	
(6)	$V_2\ P_{X3}$	→	$V_2\ X_3$	
(7)	$V_2\ X_3$	→	$V_2\ x_3\ P_{X4}$	
(8)	$V_1\ V_2\ P_{X4}$	→	$V_1\ V_2\ X_4$	
(9)	X_4	→	x_4 ...	
(I)	$V_i\ Y$	→	$Y\ V_i$, $Y \in (NT \cup T)$	Generische Regeln
(II)	$Y\ V_i$	→	$V_i\ Y$, $Y \in (NT \cup T)$	

Regelmenge 5: Datenfluss

Da alle Aktivitäten des dargestellten Prozessausschnittes auf alle Variablen zugreifen können sollen, bedeutet dies, dass die Variablen innerhalb des gesamten Wortes frei beweglich sein

müssen. Die Beweglichkeit einer Variable V_1 innerhalb eines Wortes kann durch Regeln der in Regelmenge 6 dargestellten Form modelliert werden.

$$V_1\ Y \rightarrow Y\ V_1 \quad , \quad Y \in (NT \cup T)$$
$$Y\ V_1 \rightarrow V_1\ Y \quad , \quad Y \in (NT \cup T)$$

Regelmenge 6: Beweglichkeit einer Variablen

Soll eine Variable V_i innerhalb des Wortes frei beweglich sein, so muss es für jedes Terminal und jedes Nichtterminal der Grammatik ein Regelpaar der in Regelmenge 6 dargestellten Form geben. Die Beweglichkeit von Variabeln wird also durch eine Menge von Regeln beschrieben. Diese Regelmenge wird nicht explizit aufgezählt, sondern kann generisch beschrieben werden. Dadurch ergibt sich eine Unterteilung aller WoG Regeln in explizit beschriebenen Prozessstrukturregeln auf der einen Seite und allgemein beschriebenen generischen Regeln auf der anderen Seite. Die Prozessstrukturregeln modellieren, wie ihr Name schon sagt, die Struktur des Prozesses, sie beschreiben Kontroll- und Datenfluss. Die generischen Regeln dagegen beschreiben ein allgemeines Verhalten und allgemeine Eigenschaften eines Prozesses, wie beispielsweise die Tatsache, dass Variablen von allen Aktivitäten zugreifbar sind. Die Wörter, die mit einer WoG Grammatik erzeugt werden können, werden durch die Prozessstrukturregeln beschrieben. Die generischen Regeln haben darauf keinen direkten Einfluss.

Die Unterteilung in Prozessstrukturregeln und generischen Regeln ist in Regelmenge 5 bereits vorgenommen. In dieser Arbeit werden die Prozessstrukturregeln immer mit arabischen Zahlen (1, 2, 3, 4, 5, …), die generischen Regeln immer mit römischen Zahlen beschriftet (I, II, III, IV, V, …).

3.2.5. Scope

Die Lebensdauer von Variablen wird in WoG mit Hilfe von Bereichen, sogenannten „Scopes“, modelliert. Der äußerste Scope, der den gesamten Prozess umschließt, wird Prozessscope genannt. Die Sichtbarkeit einer Variable entspricht in WoG genau ihrer Lebensdauer. Jede Variable hat einen in ihrem Scope eindeutigen Namen. Das bedeutet insbesondere, dass es auch in keinem der untergeordneten Scopes eine weitere Variable dieses Namens geben darf.

Um einen Scope S_1 zu modellieren, benötigt man zwei Prozessstrukturregeln und noch weitere generische Regeln. Regelmenge 7 zeigt diese Scoperegeln. In Regel (1) wird der Scope S_1 und die Variabeln V_1 und V_2, die in diesem Scope ihren Lebensbereich haben, erzeugt. Jeder Scope S_i wird durch eine linke und eine rechte Scopegrenze repräsentiert. Die linke Scopegrenze S_{il} stellt dabei den Beginn des Scopes und die rechte Scopegrenze S_{ir} das Ende des Scopes dar. In Regel (2) wird der Scope S_1 beendet, das bedeutet, alle in ihm lebenden Variablen sowie die Grenzen des Scopes werden gelöscht. Die generische Regel (I) sagt aus, dass die linke Grenze S_{il} eines Scopes über beendete Aktivitäten x_i nach rechts wandern kann. In den generischen Regeln (II) und (III) wird dargestellt, dass sich Variablen innerhalb ihrer Scopegrenzen frei bewegen können und dass sie ihren Scope nicht verlassen können. Die generische Regel (I) wird benötigt, um die Prozessstrukturregel (2) anwenden zu können. Da die linke Scopegrenze nur über beendete Aktivitäten nach rechts wandern kann, ist die Anwendung der Regel (2) erst dann möglich, wenn sich innerhalb des Scopes keine laufenden Aktivitäten mehr befinden. Die in Regelmenge 7 dargestellten Regeln beschreiben also, dass ein Scope so lange lebt, bis alle Aktivitäten innerhalb des Scopes beendet sind. Mit dem Ende des Scopes endet auch die Lebensdauer der Variablen dieses Scopes. Mit dem hier beschriebenen Aufbau von Scopes in WoG wird sichergestellt, dass ein Scope genau einen Eintrittspunkt und genau einen Ausrittspunkt hat. Dies gilt insbesondere für den Prozess an sich, welcher auch mit einem Scope modelliert wird.

(1) ... → S_{1l} V_1 V_2 ... S_{1r}
(2) S_{1l} V_1 V_2 S_{1r} → ...

(I) S_{il} x_j → x_j S_{il}
(II) $V_{i,Sj}$ Y → Y $V_{i,Sj}$, Y ¬∈ { S_{jl} , S_{jr}}
(III) Y $V_{i,Sj}$ → $V_{i,Sj}$ Y , Y ¬∈ { S_{jl} , S_{jr}}

Regelmenge 7: Scope

In Abbildung 14 wird ein Beispielprozess mit zwei in sich verschachtelten Scopes S_1 und S_2 dargestellt. Die Variablen V_1 und V_2 leben in dem Scope S_1 und die Aktivitäten X_1, X_2, X_3 und X_4 können auf diese Variablen zugreifen. Im Scope S_2 leben die Variablen V_3 und V_4 und nur die Aktivitäten X_2 und X_3 können auf V_3 und V_4 zugreifen. Der Zugriff von Aktivitäten auf Variablen wird in diesem Beispiel aus Gründen der Übersichtlichkeit nicht modelliert. Ein ausführliches Beispiel wird in Abbildung 15 und Regelmenge 9 dargestellt.

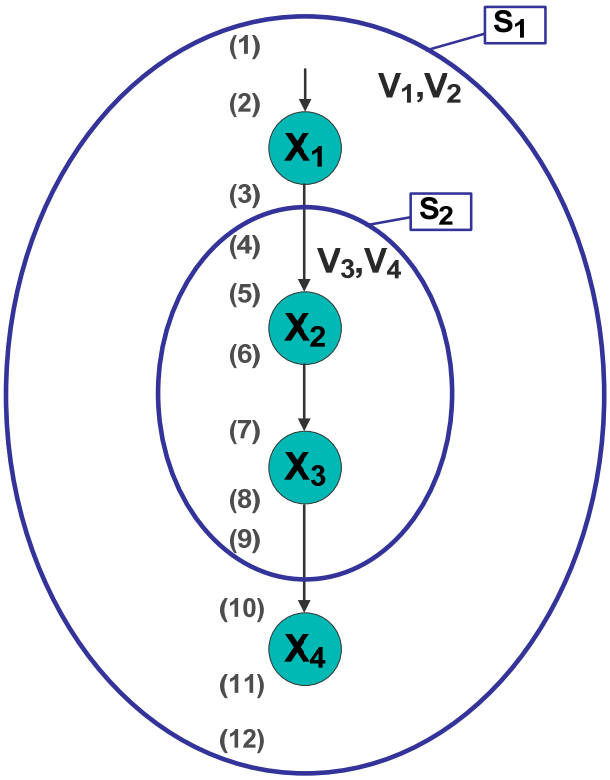


Abbildung 14: Verschachtelte Scopes

In WoG wird dieser Prozess in Regelmenge 8 dargestellt. Regel (1) zeigt die Erzeugung des Prozessscopes S_1 und der in ihm lebenden Variablen V_1 und V_2. Gleichzeitig wird auch der Platzhalter P_{X1} erzeugt. Mit diesem Platzhalter wird modelliert, dass der Kontrollfluss innerhalb des Prozessscopes S_1 bei Aktivität X_1 beginnt. Regel (2) beschreibt dann die Aktivierung der Aktivität X_1, nachdem der Scope und die Variablen erzeugt wurden. In Regel (3) wird die Aktivität X_1 beendet und der Platzhalter für den Scope S_2 erzeugt. Regel (4) beschreibt die Erzeugung des Scope S_2 und der in ihm lebenden Variablen V_3 und V_4 sowie des Platzhalters P_{X2}. In Regel (5) wird dann mit Hilfe des Platzhalters P_{X2} die Aktivität X_2 aktiviert. Die Regeln (6) bis (8) beschreiben die Abarbeitung der Sequenz von X_2 und X_3 innerhalb des Scopes S_2. Regel (9) modelliert die Beendigung des Scopes S_2, also das Löschen der Scopegrenzen und der in ihm lebenden

Variablen V_3 und V_4. Da der Kontrollfluss nach dem Scope S_2 bei Aktivität X_4 weitergeht, wird in dieser Regel gleichzeitig der Platzhalter P_{X4} erzeugt. In Regel (10) wird mit Hilfe dieses Platzhalters die Aktivität X_4 aktiviert. Regel (11) beendet die Aktivität X_4 und die Regel (12) beendet den Scope S_1. Da mit diesem Scope der gesamte Prozess beendet wird, wird in Regel (12) kein weiterer Platzhalter erzeugt. Die generischen Regeln (I) bis (III) wurden schon in der Regelmenge 7 ausführlich beschrieben.

(1)	S	$\rightarrow$	$S_{1l}\ V_1\ V_2\ P_{X1}\ S_{1r}$
(2)	P_{X1}	$\rightarrow$	X_1
(3)	X_1	$\rightarrow$	$x_1\ P_{S2}$
(4)	P_{S2}	$\rightarrow$	$S_{2l}\ V_3\ V_4\ P_{X2}\ S_{2r}$
(5)	P_{X2}	$\rightarrow$	X_2
(6)	X_2	$\rightarrow$	$x_2\ P_{X3}$
(7)	P_{X3}	$\rightarrow$	X_3
(8)	X_3	$\rightarrow$	x_3
(9)	$S_{2l}\ V_3\ V_4\ S_{2r}$	$\rightarrow$	P_{X4}
(10)	P_{X4}	$\rightarrow$	X_4
(11)	X_4	$\rightarrow$	x_4
(12)	$S_{1l}\ V_1\ V_2\ S_{1r}$	$\rightarrow$	ε
(I)	$S_{il}\ x_j$	$\rightarrow$	$x_j\ S_{il}$
(II)	$V_{i,Sj}\ Y$	$\rightarrow$	$Y\ V_{i,Sj}\ , Y \neg\in \{ S_{jl} , S_{jr}\}$
(III)	$Y\ V_{i,Sj}$	$\rightarrow$	$V_{i,Sj}\ Y\ , Y \neg\in \{ S_{jl} , S_{jr}\}$

Regelmenge 8: Prozess mit verschachtelten Scopes

Die Anwendung der Regeln aus der Regelmenge 8 ist in Ableitung 2 dargestellt. Das Beenden des Prozessscopes geschieht in den beiden letzten Ableitungsschritten. Sobald sich zwischen den beiden Scopegrenzen S_{1l} und S_{1r} nur noch Variablen und beendete Aktivitäten befinden, können die Scopegrenzen zusammengeschoben werden. Erst dann können der Scope und die in ihm lebenden Variablen gelöscht werden.

Im Allgemeinen gibt es für jedes produzierte Wort mehrere Ableitungen. Nach dem im dritten Ableitungsschritt mit der Regel (3) die Aktivität X_1 beendet wurde, wird in nächsten Schritt mit Regel (4) der Scope S_2 aktiviert. Alternativ dazu hätte man auch die generischen Regeln (II) und (I) anwenden können um die Variablen V_1 und V_2 und die linke Scopegrenze S_{1l} nach rechts über die beendete Aktivität x_1 zu schieben.

(1)
$S \rightarrow S_{1l}\ V_1\ V_2\ \underline{P_{X1}}\ S_{1r}$

(2)
$\rightarrow S_{1l}\ V_1\ V_2\ \underline{X_1}\ S_{1r}$

(3)
$\rightarrow S_{1l}\ V_1\ V_2\ x_1\ \underline{P_{S2}}\ S_{1r}$

(4)
$\rightarrow S_{1l}\ V_1\ V_2\ x_1\ S_{2l}\ V_3\ V_4\ \underline{P_{X2}}\ S_{2r}\ S_{1r}$

(5)
$\rightarrow S_{1l}\ V_1\ V_2\ x_1\ S_{2l}\ V_3\ V_4\ \underline{X_2}\ S_{2r}\ S_{1r}$

(6)
$\rightarrow S_{1l}\ V_1\ V_2\ x_1\ S_{2l}\ x_2\ \underline{P_{X3}}\ S_{2r}\ S_{1r}$

(7)
$\rightarrow S_{1l}\ V_1\ V_2\ x_1\ S_{2l}\ V_3\ V_4\ x_2\ \underline{X_3}\ S_{2r}\ S_{1r}$

(8)
$\rightarrow S_{1l}\ V_1\ V_2\ x_1\ \underline{S_{2l}}\ \underline{V_3}\ \underline{V_4}\ x_2\ x_3\ S_{2r}\ S_{1r}$

(II)*, (I)*
$\rightarrow S_{1l}\ V_1\ V_2\ x_1\ x_2\ x_3\ \underline{S_{2l}}\ \underline{V_3}\ \underline{V_4}\ \underline{S_{2r}}\ S_{1r}$

(9)
$\rightarrow S_{1l}\ V_1\ V_2\ x_1\ x_2\ x_3\ \underline{P_{X4}}\ S_{1r}$

(10)
$\rightarrow S_{1l}\ V_1\ V_2\ x_1\ x_2\ x_3\ \underline{X_4}\ S_{1r}$

(11)
$\rightarrow \underline{S_{1l}}\ \underline{V_1}\ \underline{V_2}\ x_1\ x_2\ x_3\ x_4\ S_{1r}$

(II)*, (I)*
$\rightarrow x_1\ x_2\ x_3\ x_4\ \underline{S_{1l}}\ \underline{V_1}\ \underline{V_2}\ \underline{S_{1r}}$

(12)
$\rightarrow x_1\ x_2\ x_3\ x_4$

Ableitung 2: Verschachtelter Scope

Abbildung 15 zeigt den zuvor vorgestellten Prozess, erweitert durch lesende und schreibende Variablenzugriffe. Die Aktivierungs- und Beendigungsregeln der Aktivitäten werden um die entsprechenden Variablen ergänzt. Diese Regeln sind in Regelmenge 9 dargestellt.

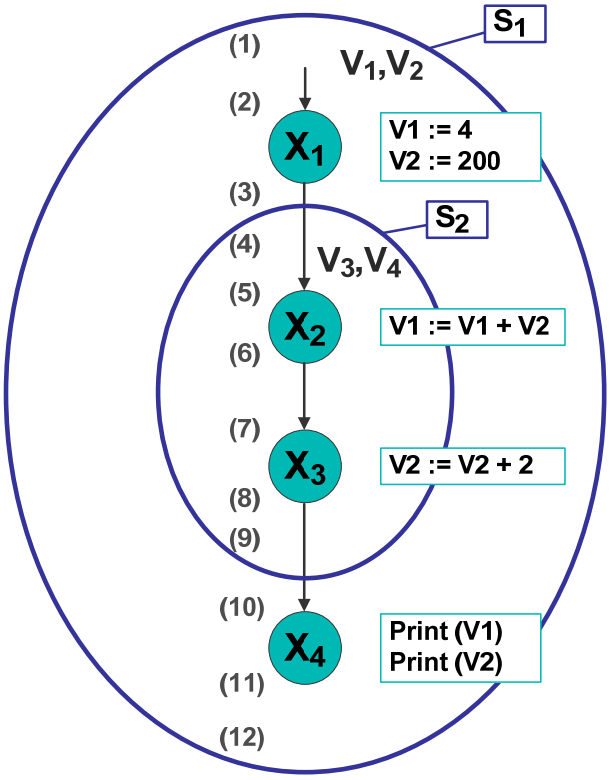


Abbildung 15: Scopes und Datenfluss

(1)	S	→	$S_{1l}\ V_1\ V_2\ P_{X1}\ S_{1r}$
(2)	P_{X1}	→	X_1
(3)	$V_1\ V_2\ X_1$	→	$V_1\ V_2\ x_1\ P_{S2}$
(4)	P_{S2}	→	$S_{2l}\ V_3\ V_4\ P_{X2}\ S_{2r}$
(5)	$V_1\ V_2\ P_{X2}$	→	$V_1\ V_2\ X_2$
(6)	$V_3\ X_2$	→	$V_3\ x_2\ P_{X3}$
(7)	$V_2\ P_{X3}$	→	$V_2\ X_3$
(8)	$V_4\ X_3$	→	$V_4\ x_3$
(9)	$S_{2l}\ V_3\ V_4\ S_{2r}$	→	P_{X4}
(10)	$V_1\ V_2\ P_{X4}$	→	$V_1\ V_2\ X_4$
(11)	X_4	→	x_4
(12)	$S_{1l}\ V_1\ V_2\ S_{1r}$	→	ε
(I)	$S_{il}\ x_j$	→	$x_j\ S_{il}$
(II)	$V_{i,Sj}\ Y$	→	$Y\ V_{i,Sj}\ , Y \neg\in \{S_{jl}, S_{jr}\}$
(III)	$Y\ V_{i,Sj}$	→	$V_{i,Sj}\ Y\ , Y \neg\in \{S_{jl}, S_{jr}\}$

Regelmenge 9: Prozess mit verschachtelten Scopes und Datenfluss

3.2.6. Verzweigung (Fork)

Bisher wurde der Kontrollfluss immer als Sequenz dargestellt. Der Kontrollfluss eines Prozessmodells kann aber im allgemeinen Fall auch Verzweigungen enthalten. Dadurch ist es unter anderem möglich, parallelen Kontrollfluss zu modellieren. Aktivitäten, die parallel ablaufen, besitzen untereinander keine Kontrollflussabhängigkeiten. Eine Verzweigung bedeutet, dass die Aktivität, von der die Verzweigung ausgeht, mehr als eine Folgeaktivität hat.

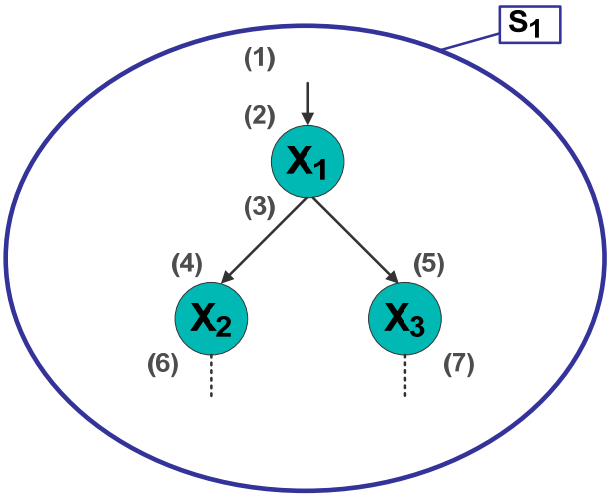


Abbildung 16: Verzweigung

Abbildung 16 zeigt einen Prozessausschnitt mit einer Verzweigung. Nachdem die Aktivität X_1 beendet wird, geht der Kontrollfluss zu den beiden Folgeaktivitäten X_2 und X_3. Diese Aktivitäten

können nun parallel abgearbeitet werden. Regelmenge 10 modelliert diesen Prozessausschnitt in WoG.

(1)	S	→	$S_{1l}\ P_{X1}\ S_{1r}$
(2)	P_{X1}	→	X_1
(3)	X_1	→	$x_1\ P_{X2}\ P_{X3}$
(4)	P_{X2}	→	X_2
(5)	P_{X3}	→	X_3
(6)	X_2	→	$x_2 \ldots$
(7)	X_3	→	$x_3 \ldots$
	...	→	...
(I)	$S_{il}\ x_j$	→	$x_j\ S_{il}$
(II)	$V_{i,Sj}\ Y$	→	$Y\ V_{i,Sj}\ , Y \neg\in \{S_{jl}, S_{jr}\}$
(III)	$Y\ V_{i,Sj}$	→	$V_{i,Sj}\ Y\ , Y \neg\in \{S_{jl}, S_{jr}\}$

Regelmenge 10: Verzweigung

In Regel (3) wird die Aktivität X_1 beendet und die Platzhalter für die Folgeaktivitäten X_2 und X_3 werden aktiviert. Dies bedeutet, dass im nächsten Schritt entweder X_2 (Regel (4)) oder X_3 (Regel (5)) aktiviert werden können, beides ist möglich. Eine Parallelität, wie sie in Abbildung 16 dargestellt ist, bedeutet nicht, dass X_2 und X_2 gleichzeitig aktiviert werden. Die Aktivierung der Aktivitäten erfolgt immer nacheinander. Die Ausführung der Aktivitäten kann jedoch parallel erfolgen. Diese Tatsache ist in Abbildung 17 und Abbildung 18 dargestellt. Das Prozessmodell fordert die Bedingung $t_0 < t_1$ und $t_0 < t_2$, dies ist durch den Kontrollfluss vorgegeben. Damit sind zwei unterschiedliche Aktivierungsreihenfolgen möglich. In Abbildung 17 wird X_2 zuerst aktiviert, danach X_3. Hier gilt, dass $t_1 < t_2$ ist, da die Aktivierung immer nacheinander erfolgt. Im Zeitraum zwischen t_2 und t_3 werden die beiden Aktivitäten X_2 und X_3 parallel ausgeführt. Die zweite mögliche Aktivierungsreihenfolge wird in Abbildung 18 dargestellt.

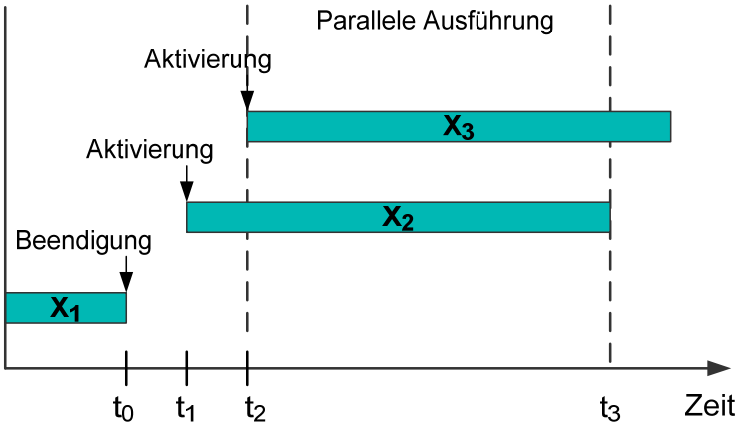


Abbildung 17: Aktivierungsreihenfolge und parallele Ausführung (a)

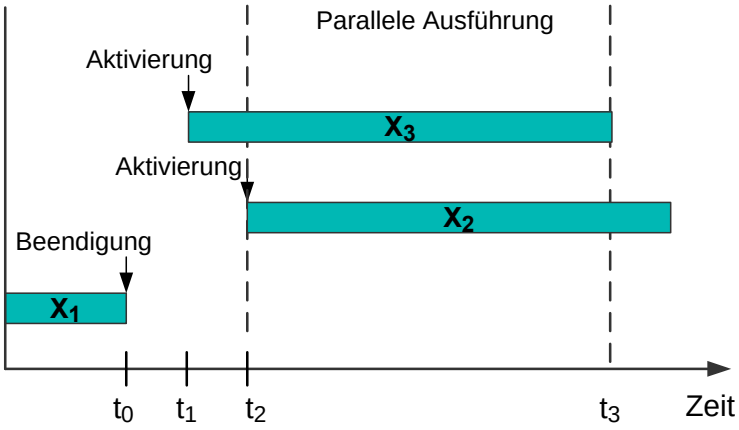


Abbildung 18: Aktivierungsreihenfolge und parallele Ausführung (b)

Die konkrete Reihenfolge der Aktivierung von parallelen Aktivitäten ist, wie im vorigen Absatz bereits erwähnt, durch das Prozessmodell nicht vorgegeben. Sie wird durch die Laufzeitumgebung bestimmt, in welcher der Prozess ausgeführt wird.

3.2.6.1. Parallelität und Datenfluss

Parallelität bedeutet Unabhängigkeit vom Kontrollfluss. Es gibt Sprachen, in denen zusätzlich Unabhängigkeit des Datenflusses gefordert wird. Wenn zwei Aktivitäten keine Kontrollflussabhängigkeit haben, dann dürfen sie auch keine Datenflussabhängigkeit haben („Kein Datenfluss ohne Kontrollfluss“). Diese Forderung wird auch als „Bernstein Bedingung“ bezeichnet [DRV00] und wird zum Beispiel im graphbasierten Metamodell verlangt, welches in [LR00] vorgestellt wird.

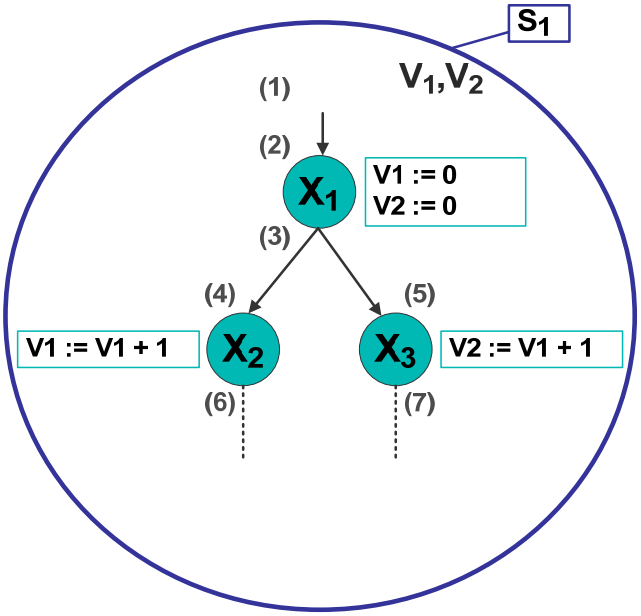


Abbildung 19: Datenfluss ohne Kontrollfluss

Abbildung 19 zeigt ein Beispielprozess für einen Datenfluss ohne Kontrollfluss. Die Aktivitäten X_2 und X_3 befinden sich in parallelen Kontrollflüssen, sie sind also voneinander unabhängig bezüglich des Kontrollflusses. Sowohl Aktivität X_2 als auch Aktivität X_3 greifen auf die Variable V_1 zu. X_2 liest und schreibt auf V_1 und X_3 liest V_1. Somit sind diese beiden Aktivitäten bezüglich des Datenflusses nicht unabhängig voneinander. Das Problem bei einer solchen Modellierung liegt darin, dass zwei

Prozessdurchläufe mit den gleichen Ausgangswerten unterschiedliche Ergebnisse liefern können. Betrachtet man einen Durchlauf des Prozesses, bei dem die Aktivität X_2 schnell und die Aktivität X_3 langsam durchläuft, dann liest X_3 den Wert der Variablen V_1 nachdem X_2 diese Variable beschrieben hat. Dieser Fall ist in Tabelle 1 in der Spalte „Durchlauf 1“ dargestellt. Betrachtet man dagegen einen Durchlauf des Prozesses, bei dem die Aktivität X_3 schnell und die Aktivität X_2 langsam läuft, dann liest X_3 den Wert der Variablen V_1 bevor X_2 diese Variable beschreibt. Dies ist in der Tabelle 1 in der Spalte „Durchlauf 2“ dargestellt. Vergleicht man beide Durchläufe, so stellt man fest, dass der Wert der Variablen V_2 am Ende des Prozesses in jeden Durchlauf einen anderen Wert hat. So ein Verhalten ist in den meisten Fällen unerwünscht. Es wird erwartet, dass zwei Prozessdurchläufe mit den gleichen Ausgangswerten die gleichen Ergebnisse produzieren.

	Durchlauf 1	**Durchlauf 2**
Deklaration	V_1 = undef , V_2 = undef	V_1 = undef , V_2 = undef
Aktion	X_1: V_1 := 0 ; V_2 := 0	X_1: V_1 := 0 ; V_2 := 0
Aktuelle Werte	V_1 = 0 , V_2 = 0	V_1 = 0 , V_2 = 0
Aktion	X_2: V_1 := V_1 + 1	X_3: V_2 := V_1 + 1
Aktuelle Werte	V_1 = 1 , V_2 = 0	V_1 = 0 , V_2 = 1
Aktion	X_3: V_2 := V_1 + 1	X_2: V_1 := V_1 + 1
Ergebniswerte	V_1 = 1 , **V_2 = 2**	V_1 = 1 , **V_2 = 1**

Tabelle 1: Prozessdurchläufe mit gleichen Ausgangswerten

Da WoG aber so entworfen werden soll, dass möglichst viele Workflowsprachen auf WoG abgebildet werden können, bezieht sich Parallelität in WoG nur auf die Unabhängigkeit des Kontrollflusses. WoG erlaubt die Modellierung von Datenfluss ohne einen entsprechenden Kontrollfluss. Variablen leben innerhalb ihres Scopes, sie sind dort frei beweglich und für alle Aktivitäten innerhalb dieses Scopes sichtbar. Damit sind sie unabhängig vom konkreten Kontrollfluss innerhalb dieses Scopes. Der Modellierer ist verantwortlich für eine „gute“ Modellierung.

3.2.7. Zusammenführung (Join)

In einem Prozessmodell können parallel verlaufende Kontrollflüsse zusammengeführt werden. Diese Zusammenführung nennt man auch „Join“. Eine Zusammenführung bedeutet, dass die Aktivität, an der die Zusammenführung stattfindet, mehr als eine direkte Vorgängeraktivität hat.

Abbildung 20 zeigt einen Prozessausschnitt mit einer Zusammenführung. Die Aktivitäten X_2 und X_3 werden in parallelen Kontrollflüssen ausgeführt. Nachdem beide Aktivitäten beendet worden sind, sollen die parallelen Kontrollflüsse zusammengeführt werden. Die Aktivität X_4 ist eine gemeinsame Folgeaktivität von X_2 und X_3. Sie darf erst dann aktiviert werden, wenn die Aktivitäten X_2 und X_3 beendet worden sind. Regelmenge 11 modelliert diesen Prozessausschnitt in WoG. Die Platzhalter, welche die Aktivitäten bei ihrer Beendigung erzeugen, sind in Abbildung 20 durch entsprechend beschriftete kleine Kreise dargestellt.

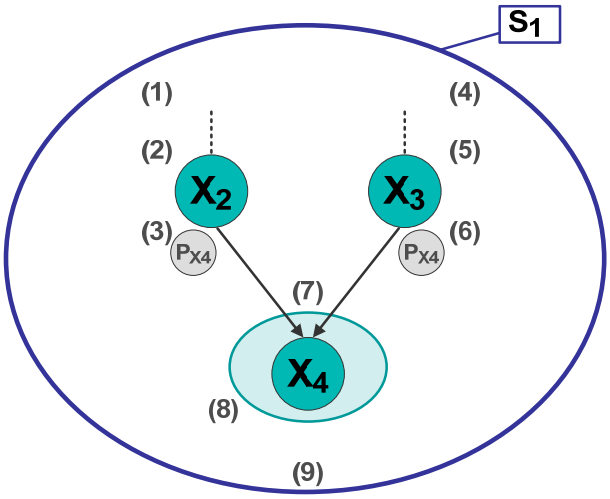


Abbildung 20: Zusammenführung

(1)	...	→	P_{X2}
(2)	P_{X2}	→	X_2
(3)	X_2	→	$x_2 P_{X4}$
(4)	...	→	P_{X3}
(5)	P_{X3}	→	X_3
(6)	X_3	→	$x_3 P_{X4}$
(7)	$P_{X4} P_{X4}$	→	X_4
(8)	X_4	→	x_4
(9)	$S_{1l} S_{1r}$	→	ε
(I)	$S_{il} x_j$	→	$x_j S_{il}$
(II)	$V_{i,Sj} Y$	→	$Y V_{i,Sj}$, $Y \neg\in \{S_{jl}, S_{jr}\}$
(III)	$Y V_{i,Sj}$	→	$V_{i,Sj} Y$, $Y \neg\in \{S_{jl}, S_{jr}\}$
(IV)	$P_{Xi} Y$	→	$Y P_{Xi}$, $Y \in NT \cup T$
(V)	$Y P_{Xi}$	→	$P_{Xi} Y$, $Y \in NT \cup T$

Regelmenge 11: Zusammenführung

In Regel (3) der Regelmenge 11 wird die Aktivität X_2 beendet und ein Platzhalter P_{X4} für die Folgeaktivität X_4 erzeugt. Regel (6) beendet die Aktivität X_3 und erzeugt ebenfalls einen Platzhalter P_{X4} für die Folgeaktivität X_4. Die Aktivierungsregel für die Aktivität X_4 ist in Regel (7) dargestellt. Die Anzahl der Platzhalter P_{X4} auf der linken Regelseite entspricht der Anzahl der in Abbildung 20 dargestellten eingehenden Kanten der Aktivität X_4. Dies bedeutet, dass die Aktivität X_4 erst dann aktiviert werden kann, wenn alle Vorgängeraktivitäten von X_4 beendet worden sind. Damit die Regel (7) angewendet werden kann, müssen zuvor die beiden Platzhalter P_{X4} zusammengeschoben werden. Dies wird durch die generischen Regeln (IV) und (V) ermöglicht. Sie sagen aus, dass sich Platzhalter im Wort frei bewegen können. Die Tatsache, dass sich Platzhalter im Wort frei bewegen können, bedeutet unter anderem, dass die Aktivität, an der die Zusammenführung stattfindet (in unserem Beispiel X_4), an einer beliebigen Position im Wort aktiviert werden kann. Dadurch ist es möglich, dass zwei gleiche Prozessdurchläufe

unterschiedliche Wörter erzeugen. Abbildung 21 zeigt drei verschiedene Ableitungen für den gleichen Prozessdurchlauf. Die Ausgangsposition (Punkt 1) zeigt immer das gleiche Teilwort. Die Aktivitäten X_2 und X_3 wurden beendet und die entsprechen Platzhalter P_{X4} wurden erzeugt. Im nächsten Schritt (Punkt 2) werden die Platzhalter verschoben. Sie werden in jeder der drei Ableitungen an unterschiedliche Positionen bewegt. Im letzten dargestellten Schritt (Punkt 3) wird die Aktivität X_4 aktiviert. Die so entstandenen Wörter sind in jeder Ableitung unterschiedlich, obwohl die Aktivierungsreihenfolge der Aktivitäten immer gleich war. Die generischen Regeln (IV) und (V) können also die Struktur des Ergebniswortes verändern.

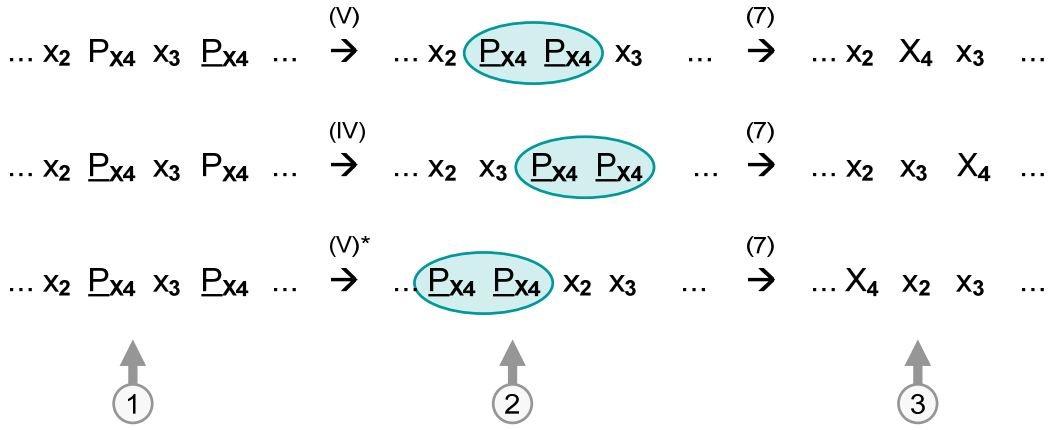


Abbildung 21: gleiche Prozessdurchläufe – unterschiedliche Wörter

So ein Verhalten ist in WoG nicht erwünscht. Gleiche Prozessdurchläufe sollen die gleichen Wörter produzieren. Um dies zu erreichen, werden neue Platzhalter P_{Xim} („m“ steht für mobil) eingeführt und die entsprechenden generischen Regeln modifiziert. Für diese neuen Platzhalter P_{Xim} gelten die bisherigen generischen Regeln, welche beschreiben, dass diese Platzhalter im Wort frei beweglich sind. Alle anderen Platzhalter P_{Xi} sind dagegen nicht mehr beweglich. Bei einer Zusammenführung mehrerer paralleler Kontrollflüsse erzeugen die Vorgängeraktivitäten mobile Platzhalter. Genau eine der Vorgängeraktivitäten erzeugt einen nicht mobilen Platzhalter. Ohne Beschränkung der Allgemeinheit wählen wir dafür immer die letzte (in der Abbildung immer die rechte) Vorgängeraktivität. Die Aktivierung einer Aktivität ist somit immer nur an der Position dieses einen festen Platzhalters möglich.

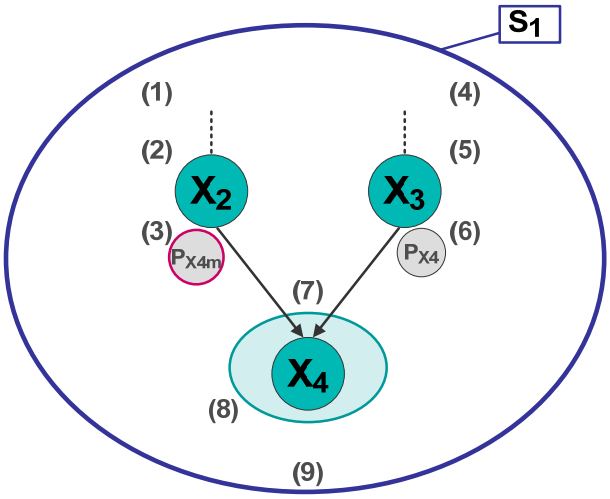


Abbildung 22: Zusammenführung – Mobile Platzhalter

Wie in Abbildung 22 zu sehen ist, wird bei Beendigung der Aktivität X_2 ein mobiler Platzhalter P_{X4m} für die Folgeaktivität X_4 erzeugt. Bei Beendigung der Aktivität X_3 wird ebenfalls ein Platzhalter für

die Folgeaktivität X_4 erzeugt. Da X_3 aber die „letzte“ Vorgängeraktivität von X_4 ist, ist dieser erzeugte Platzhalter P_{X4} nicht mobil. In WoG wird dies in Regel (3) und Regel (6) der Regelmenge 12 modelliert. Regel (7) zeigt die Aktivierungsregel der Aktivität X_4. Auf der linken Regelseite befinden sich die zwei zuvor erzeugten Platzhalter P_{X4m} und P_{X4}. Die modifizierten generischen Regeln (IV) und (V) sagen aus, dass sich nur noch der mobile Platzhalter frei im Wort bewegen kann.

(1)	$\dots$	$\rightarrow$	P_{X2}
(2)	P_{X2}	$\rightarrow$	X_2
(3)	X_2	$\rightarrow$	$x_2\ P_{X4m}$
(4)	$\dots$	$\rightarrow$	P_{X3}
(5)	P_{X3}	$\rightarrow$	X_3
(6)	X_3	$\rightarrow$	$x_3\ P_{X4}$
(7)	$P_{X4m}\ P_{X4}$	$\rightarrow$	X_4
(8)	X_4	$\rightarrow$	x_4
(9)	$S_{1l}\ S_{1r}$	$\rightarrow$	ε
(I)	$S_{il}\ x_j$	$\rightarrow$	$x_j\ S_{il}$
(II)	$V_{i,Sj}\ Y$	$\rightarrow$	$Y\ V_{i,Sj}\ ,\ Y \neg\in \{ S_{jl}, S_{jr}\}$
(III)	$Y\ V_{i,Sj}$	$\rightarrow$	$V_{i,Sj}\ Y\ ,\ Y \neg\in \{ S_{jl}, S_{jr}\}$
(IV)	$P_{Xim}\ Y$	$\rightarrow$	$Y\ P_{Xim}\ ,\ Y \in NT \cup T$
(V)	$Y\ P_{Xim}$	$\rightarrow$	$P_{Xim}\ Y\ ,\ Y \in NT \cup T$

Regelmenge 12: Zusammenführung – mobiler Platzhalter

3.2.8. Verzweigung und Zusammenführung

Die Konstrukte Verzweigung und Zusammenführung werden oft gemeinsam verwendet. Die Verzweigung erlaubt es, den Kontrollfluss in mehrere parallele Stränge aufzusplitten. Um diese parallelen Stränge wieder zu vereinen, wird das Konstrukt Zusammenführung verwendet.

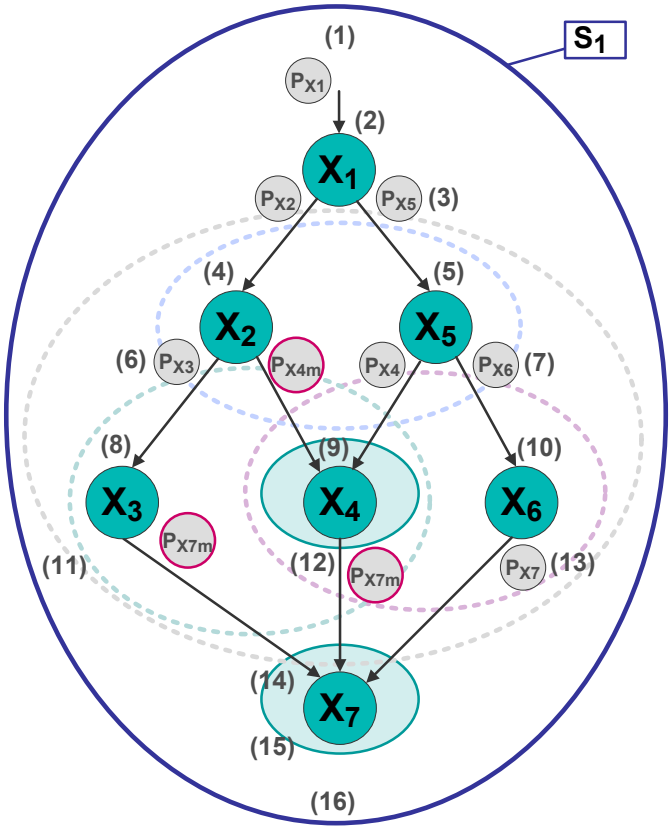
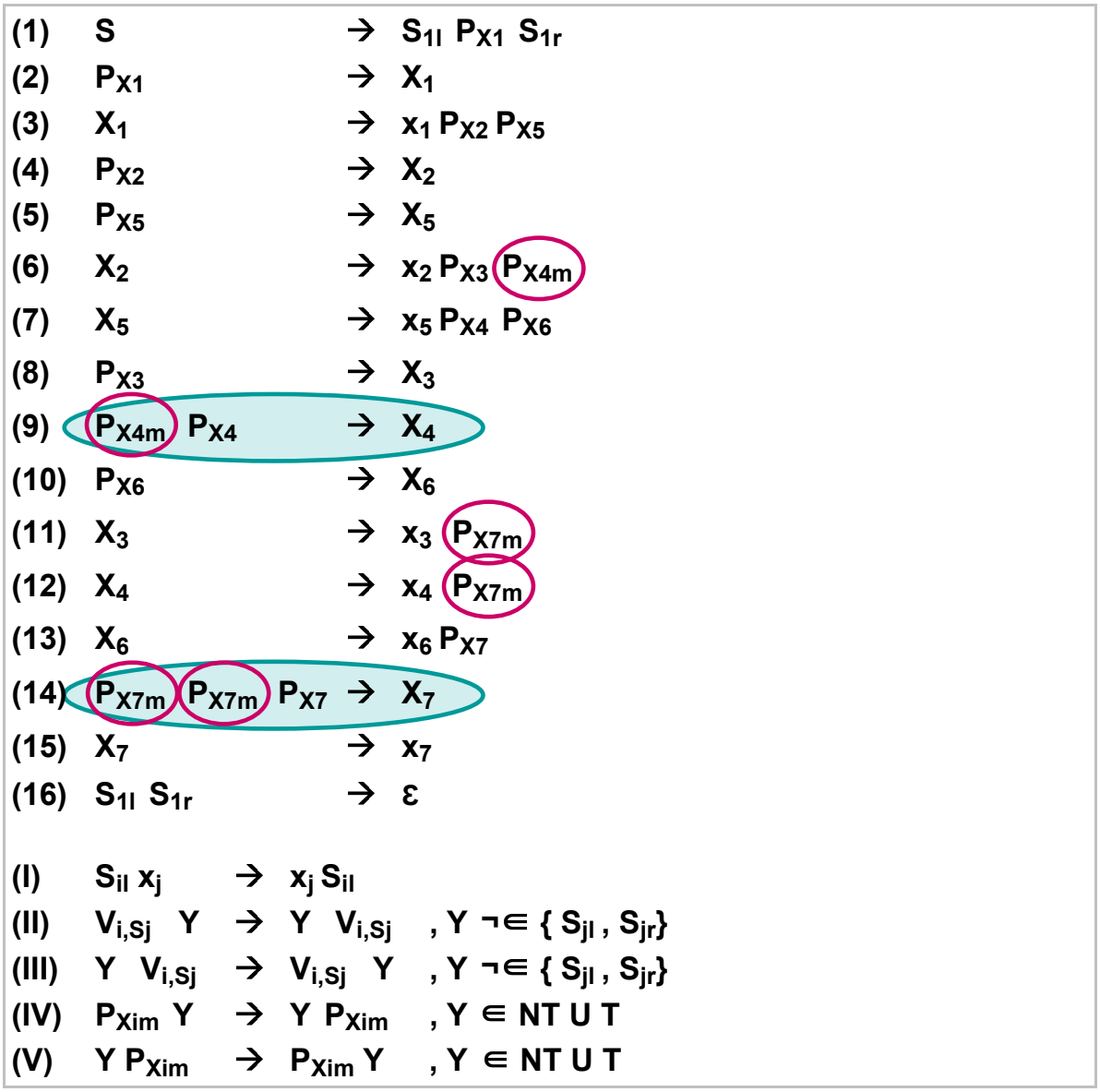


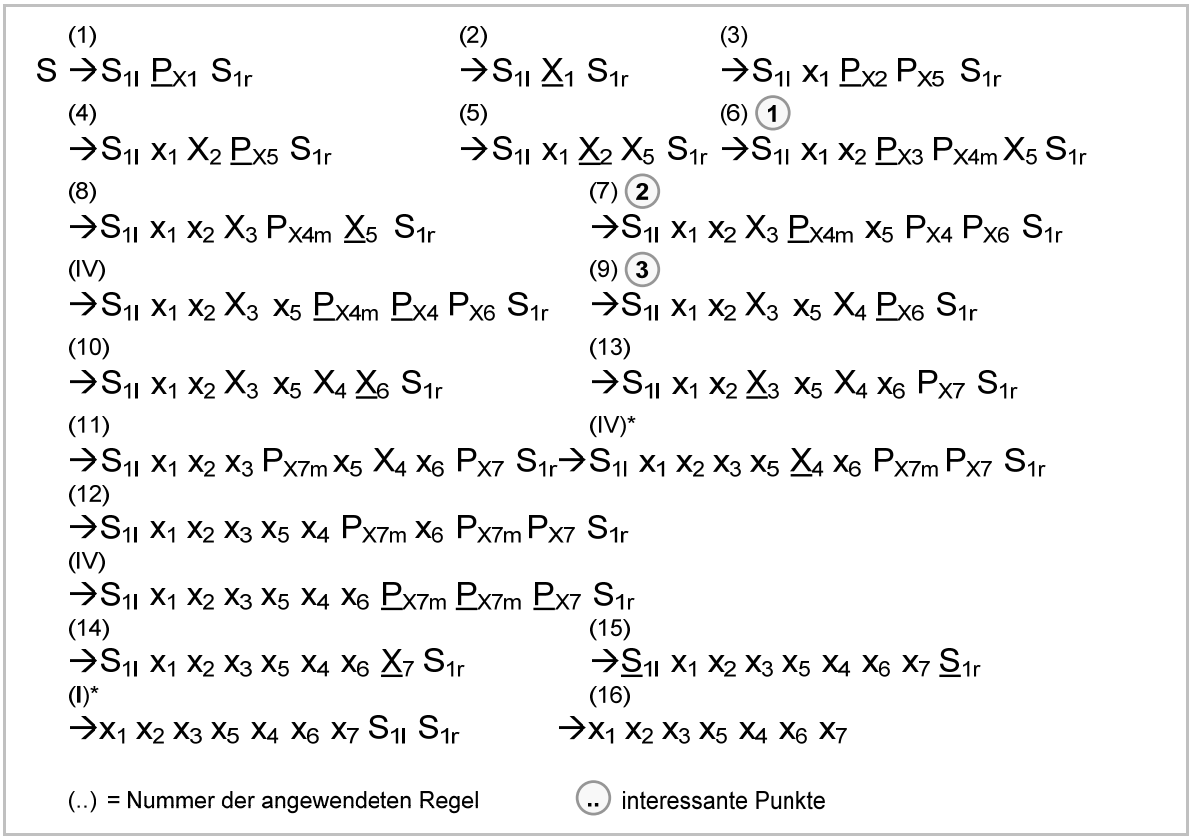
Abbildung 23: Verschachtelte und sich überlappende Parallelitäten

Die gemeinsame Verwendung von Verzweigungen und Zusammenführungen wird im Folgenden an einem etwas komplexeren Beispiel dargestellt. Abbildung 23 zeigt den Beispielprozess grafisch, Regelmenge 13 beschreibt den gleichen Prozess in WoG. Obwohl der betrachtete Prozess verschachtelte und sich überlappende Parallelitäten enthält, können die entsprechenden Regeln nach einem einfachen Schema aufgebaut werden. Die Aktivierungsregel einer Aktivität wird bestimmt durch die Anzahl der direkten Vorgängeraktivitäten. Die Beendigungsregel wird bestimmt durch die direkten Nachfolgeraktivitäten. Um beispielsweise die Aktivierungs- und Beendigungsregel für die Aktivität X_4 aufzubauen, sind lediglich die Aktivitäten X_2, X_5 und X_7 relevant. Man benötigt kein weiteres Wissen über den restlichen Prozess. Die Aktivität X_4 besitzt genau zwei direkte Vorgängeraktivitäten. Dies bedeutet, dass die Aktivierungsregel von X_4 auf der linken Seite zwei Platzhalter P_{X4} enthalten muss. Genau einer der Platzhalter ist fest, alle anderen sind mobil. Nach diesem Muster ist Regel (9) aufgebaut. Die Aktivität X_4 besitzt genau eine direkte Nachfolgeraktivität, die Aktivität X_7. Die Beendigungsregel von X_4 muss also genau einen Platzhalter erzeugen, in diesem Fall einen Platzhalter für die Aktivität X_7. Da X_4 nicht die letzte direkte Vorgängeraktivität von X_7 ist, wird ein mobiler Platzhalter für X_7 erzeugt. Regel (12) ist die entsprechende Beendigungsregel für X_4.



Regelmenge 13: Verschachtelte und sich überlappende Parallelitäten

Ableitung 3 beschreibt eine mögliche Ableitung für diesen Beispielprozess. Punkt 1 zeigt, wie mit Regel (6) die Aktivität X_2 beendet wird und die Platzhalter P_{X3} und P_{X4m} für die direkten Folgeaktivitäten X_3 und X_4 erzeugt werden. Da X_3 nur eine Vorgängeraktivität hat (die Aktivität X_2), genügt dieser eine Platzhalter P_{X3}, um sie zu aktivieren. Dies geschieht im nächsten Ableitungsschritt mit der Regel (8). X_4 dagegen hat zwei direkte Vorgängeraktivitäten, benötigt also auch zwei Platzhalter P_{X4} für die Aktivierung. Bis Punkt 1 wurde erst ein Platzhalter P_{X4m} erzeugt, daher kann an dieser Stelle noch keine Aktivierung von X_4 erfolgen. Ein paar Ableitungsschritte später (Punkt 2) wird die Aktivität X_5 beendet und die Platzhalter für ihre direkten Folgeaktivitäten X_4 und X_6 erzeugt. Damit gibt es zwei Platzhalter für die Aktivität X_4. Das bedeutet, dass alle Vorgängeraktivitäten von X_4 beendet worden sind und die Aktivität X_4 aktiviert werden kann. Zuvor muss aber noch der mobile Platzhalter P_{X4m} nach rechts zum Platzhalter P_{X4} geschoben werden. Punkt 3 zeigt, wie mit Regel (9) die Aktivität X_4 aktiviert wird.



Ableitung 3: Verschachtelte und sich überlappende Parallelitäten

3.2.9. Bedingter Kontrollfluss

Der Kontrollfluss kann von Bedingungen abhängen. Nur wenn die Bedingung erfüllt wird, fließt die Kontrolle von einer Aktivität zur anderen. In WoG gibt es drei Möglichleiten, bedingten Kontrollfluss zu modellieren. Der Kontrollfluss kann abhängen von einer booleschen Bedingung, vom Empfangen einer Nachricht oder von dem Auftreten eines Fehlers.

3.2.9.1. Kontrollfluss abhängig von Booleschen Bedingungen

In Abbildung 24 wird ein Kontrollfluss mit Boolescher Bedingung dargestellt. Wenn die Aktivität X_1 beendet ist, geht der Kontrollfluss nur dann nach X_2, wenn die Bedingung $V_1 > 10$ erfüllt ist. Dies bedeutet, dass nach dem Beenden von X_1 zunächst die Boolesche Bedingung $E_{X1,X2}$ ausgewertet werden muss. Wenn das Ergebnis dieser Auswertung der Wahrheitswert „wahr“ ist, dann wird die Aktivität X_2 aktiviert. Wenn das Ergebnis dieser Auswertung der Wahrheitswert „falsch“ ist, dann wird X_2 nicht aktiviert, der Kontrollfluss endet an dieser Stelle.

Um dieses Vorgehen in WoG zu modellieren, wird eine Aktivität vom Typ „Evaluation“ (E_i) sowie eine spezielle Variable R_j benötigt. Eine Aktivität vom Typ Evaluation wertet eine Boolesche Bedingung aus. Das Ergebnis dieser Auswertung ist ein Wahrheitswert. Um einen bedingten Kontrollfluss modellieren zu können, wird das Ergebnis nicht in einer normalen Variable V_k, sondern in einer speziellen Ergebnisvariablen R_j gespeichert. Der Unterschied dieser beiden Typen von Variablen besteht darin, dass der Wert einer normalen Variablen für die Ausführung keine Rolle spielt. Eine normale Variable ist eine Blackbox. Der Wert einer Ergebnisvariablen ist dagegen sichtbar, er wird für die Ausführung des Prozesses benötigt. Ein weiterer Unterschied ist, dass die Lebensdauer von einer normalen Variablen an einen Scope gebunden ist. Ergebnisvariablen

werden nur nach Bedarf temporär erzeugt. In Regelmenge 14 wird der bedingte Kontrollfluss des in Abbildung 24 dargestellten Beispiels modelliert.

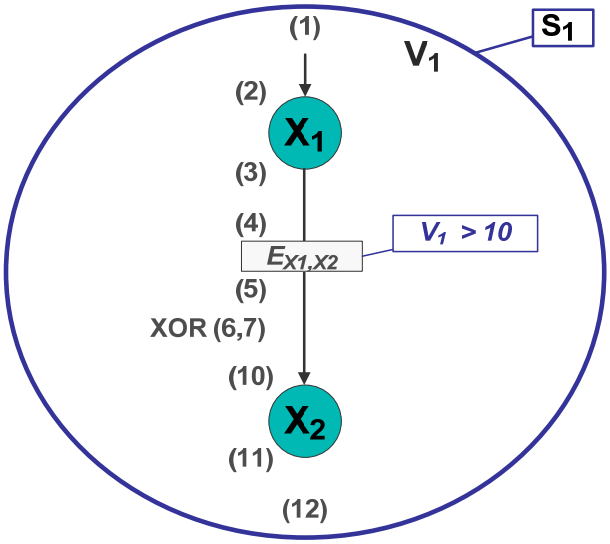


Abbildung 24: Kontrollfluss mit Boolescher Bedingung

(1)	S	→	$S_{1l}\ V_1\ P_{X1}\ S_{1r}$
(2)	P_{X1}	→	X_1
(3)	X_1	→	$x_1\ P_{E\_X1,X2}$
(4)	$V_1\ P_{E\_X1,X2}$	→	$V_1\ E_{X1,X2}$
(5)	$E_{X1,X2}$	→	$e_{X1,X2}\ R_{E\_X1,X2}$
(6)	$R_{E\_X1,X2}$	→	$T\ P_{X2}$
(7)	$R_{E\_X1,X2}$	→	F
(8)	T	→	ε
(9)	F	→	ε
(10)	P_{X2}	→	X_2
(11)	X_2	→	x_2
(12)	$S_{1l}\ V_1\ S_{1r}$	→	ε
(I)	$S_{il}\ x_j$	→	$x_j\ S_{il}$
(II)	$V_{i,Sj}\ \ Y$	→	$Y\ \ V_{i,Sj}\ \ , Y \neg\in \{ S_{jl} , S_{jr}\}$
(III)	$Y\ \ V_{i,Sj}$	→	$V_{i,Sj}\ \ Y\ \ , Y \neg\in \{ S_{jl} , S_{jr}\}$

Regelmenge 14: Kontrollfluss mit Boolescher Bedingung

Regel (3) der Regelmenge 14 beendet die Aktivität X_1 und erzeugt den Platzhalter für die Auswertung der Bedingung. In Regel (4) wird die Aktivität $E_{X1,X2}$, welche die Boolesche Bedingung auswertet, aktiviert. Sie benötigt dabei lesenden Zugriff auf alle Variablen, die in dieser Bedingung benutzt werden. Regel (5) beendet die Auswertung dieser Bedingung und speichert das Ergebnis, einen Wahrheitswert, in der Ergebnisvariable $R_{E\_X1,X2}$. Diese Variable wird erst in Regel (5) erzeugt. Regel (6) und (7) beschreiben die Auswertung des Ergebnisses der Bedingung. Auf der rechten

Seite befinden sich spezielle Nichtterminale „T“ und „F“, welche die Wahrheitswerte „wahr“ und „falsch“ („true“ und „false“) repräsentieren. Wenn zur Ausführungszeit der in $R_{E\_X1,X2}$ gespeicherte Wert „wahr“ ist, dann wird Regel (6) angewendet, da sie auf der rechten Seite das entsprechende Nichtterminal „T“ enthält. Dementsprechend wird Regel (7) angewendet, falls der Wahrheitswert der Ergebnisvariablen „falsch“ ist. Regel (6) und (7) beschreiben eigentlich einen Nichtdeterminismus, da es für das Nichtterminal $R_{E\_X1,X2}$ mehrere alternative Regeln gibt. Zur Ausführungszeit ist aber eindeutig entscheidbar, welche Regel angewendet werden soll. Die Nichtterminale „T“ und „F“ werden in Regel (8) bzw. (9) anschließend wieder gelöscht. Sie werden nur benötigt, um zur Ausführungszeit eine Entscheidung zwischen Regel (6) und (7) zu treffen.

Die Aktivität Evaluation (E_i) wertet einen booleschen Ausdruck aus und speichert das Ergebnis in einer temporären Ergebnisvariablen. Die Aktivität Assign (Ai) wertet beliebige Ausdrücke aus und speichert das Ergebnis in einer normalen Variablen ab. Das bedeutet, dass die Funktionalität einer Aktivität Ei prinzipiell auch durch die Verwendung einer Aktivität Ai modelliert werden könnte. Es ist aber eine bewusste Designentscheidung, trotzdem beide Aktivitätstypen zu verwenden. Für das Verständnis eines Prozessmodells ist es wichtig unterscheiden zu können, ob es sich bei einer Aktivität um eine beliebige Zuweisung (Ai) handelt oder ob eine Boolesche Bedingung ausgewertet wird, deren Ergebnis den Kontrollfluss beeinflusst.

3.2.9.2. Kontrollfluss abhängig von Nachrichten

Das im Folgenden beschriebene WoG Konstrukt wird motiviert durch die Workflowsprache BPEL. Die BPEL Aktivität „Pick“ repräsentiert ein polymorphes Receive [BPEL07]. Sie kann unterschiedliche Nachrichtentypen über unterschiedliche Schnittstellen empfangen.

Abbildung 25 stellt den von Nachrichten abhängigen Kontrollfluss beispielhaft dar. Die Aktivität I_1 vom Aktivitätstyp „In“ kann Nachrichten über unterschiedliche Schnittstellen empfangen. Abhängig davon, über welche Schnittstelle eine Nachricht empfangen wurde, kann sie unterschiedlich behandelt werden. Zum einen können die unterschiedlichen Schnittstellen unterschiedliche Nachrichtentypen empfangen. Es muss also möglich sein, eine Nachricht abhängig von ihrem Typ in eine entsprechende Variable abzulegen. Außerdem soll es möglich sein, den Kontrollfluss von der Schnittstelle, über die die Nachricht empfangen wurde, abhängig zu machen. Die im Beispiel dargestellten Bedingungen M1, M2 und M3 repräsentieren den Empfang einer Nachricht über unterschiedliche Schnittstellen.

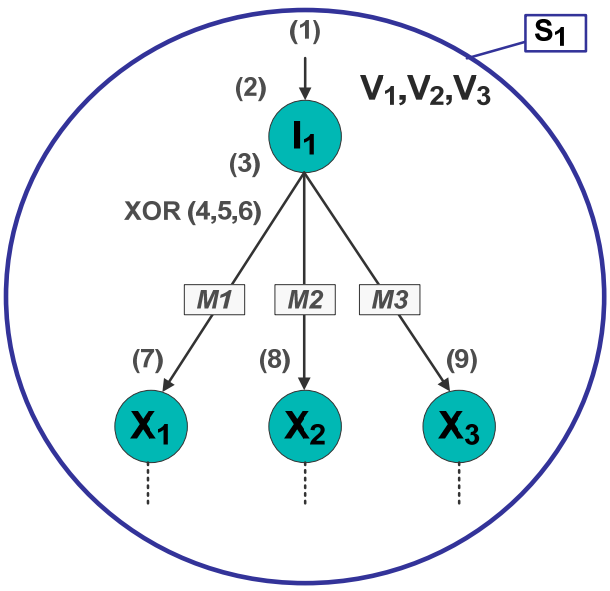


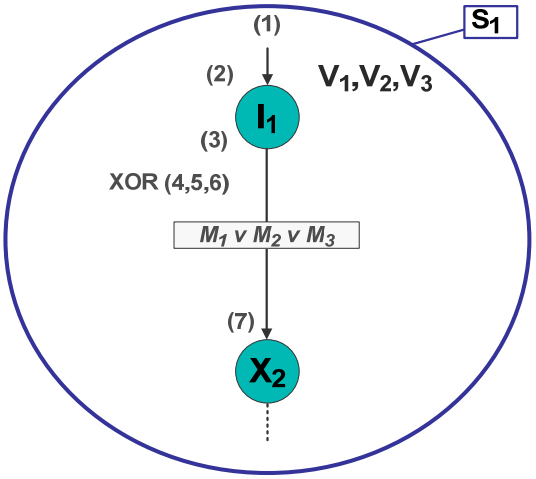
Abbildung 25: Kontrollfluss abhängig von Nachrichten

Regelmenge 15 beschreibt, wie der in Abbildung 25 dargestellte Beispielprozess in WoG modelliert wird. In Regel (1) wird der Scope S_1 mit den in ihm lebenden Variablen und der Platzhalter P_{I1} erzeugt. Für jeden unterschiedlichen Nachrichtentyp wird eine entsprechende Variable V_i erzeugt. Regel (2) aktiviert die Aktivität I_1. In Regel (3) wird I_1 beendet und es wird eine spezielle Variable R_{I1} erzeugt. In dieser temporären Variablen wird die empfangene Nachricht zwischengespeichert. Zusätzlich werden in dieser Variable noch Informationen über die Schnittstelle, über welche die Nachricht empfangen wurde, abgelegt. Regel (4), (5) und (6) beschreiben nun, dass abhängig von den Schnittstelleninformationen sowohl der Kontrollfluss als auch der Datenfluss unterschiedlich sein können (aber nicht müssen). Die Nichtterminale M_1, M_2 und M_3 repräsentieren die verschiedenen Schnittstellen, über die I_1 Nachrichten empfangen kann. Um zu bestimmen, welche der Regeln (4), (5) oder (6) zur Laufzeit angewendet wird, wird die in R_{I1} hinterlegte Schnittstelleninformation mit dem entsprechenden Nichtterminal auf der rechten Seite verglichen. Gleichzeitig wird in diesen Regeln die in R_{I1} hinterlegte empfangene Nachricht in eine normale Variable des entsprechenden Datentyps abgespeichert.

(1)	S	→	$S_{1b}\, V_1\, V_2\, V_3\, P_{I1}\, S_{1e}$
(2)	P_{I1}	→	I_1
(3)	I_1	→	$i_1\, R_{I1}$
(4)	$V_1\, R_{I1}$	→	$V_1\, M_1\, P_{X1}$
(5)	$V_2\, R_{I1}$	→	$V_2\, M_2\, P_{X2}$
(6)	$V_3\, R_{I1}$	→	$V_3\, M_3\, P_{X3}$
(7)	P_{X1}	→	X_1
(8)	P_{X2}	→	X_2
(9)	P_{X3}	→	X_3
(10)	M_1	→	ε
(11)	M_2	→	ε
(12)	M_3	→	ε
	...	→	...
(I)	$S_{il}\, x_j$	→	$x_j\, S_{il}$
(II)	$V_{i,Sj}\ Y$	→	$Y\ V_{i,Sj}\ ,\ Y \neg\in \{ S_{jl}, S_{jr}\}$
(III)	$Y\ V_{i,Sj}$	→	$V_{i,Sj}\ Y\ ,\ Y \neg\in \{ S_{jl}, S_{jr}\}$

Regelmenge 15: Kontrollfluss abhängig von Nachrichten

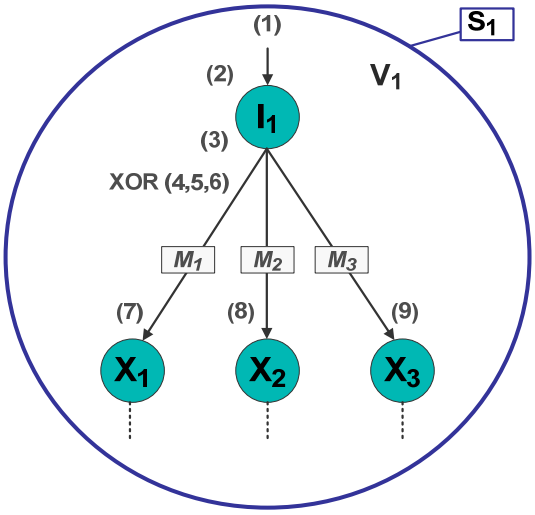
Wie bereits schon gesagt wurde, können sowohl der Kontrollfluss als auch der Datenfluss abhängig von den Schnittstelleninformationen sein, sie müssen es aber nicht. Abbildung 26 zeigt einen Beispielprozess, in dem, unabhängig davon, über welche der drei Schnittstellen eine Nachricht empfangen wurde, der Kontrollfluss immer von I_1 zu X_2 geht. Die Nachrichten, die über die drei Schnittsellen empfangen werden können, haben aber unterschiedliche Datentypen. Auf der rechten Seite der Abbildung wird der entsprechenden Ausschnitt einer WoG Regelmenge dargestellt. Wie in den Regeln (4), (5) und (6) zu sehen ist, wird immer der Platzhalter P_{X2} für die Folgeaktivität X_2 erzeugt (der Kontrollfluss ist also immer gleich). Die Regeln unterscheiden sich aber in den Variablenzugriffen (der Datenfluss ist unterschiedlich), die abhängig sind von der jeweiligen Schnittstelle.



	...	→	...
(3)	I_1	→	$i_1\ R_{I1}$
(4)	$V_1\ R_{I1}$	→	$V_1\ M_1\ P_{X2}$
(5)	$V_2\ R_{I1}$	→	$V_2\ M_2\ P_{X2}$
(6)	$V_3\ R_{I1}$	→	$V_3\ M_3\ P_{X2}$
	...	→	...

Abbildung 26: Gleicher Kontrollfluss, unterschiedliche Nachrichtentypen

Abbildung 27 zeigt einen Beispielprozess, in dem der Kontrollfluss abhängig davon ist, über welche Schnittstelle eine Nachricht empfangen wurde. In diesem Beispiel ist der Typ der Nachricht bei allen Schnittstellen gleich. Die rechte Seite der Abbildung zeigt den entsprechenden Ausschnitt einer WoG Regelmenge. Wie in den Regeln (4), (5) und (6) zu sehen ist, wird immer auf die gleiche Variable V_1 zugegriffen (der Datenfluss ist also gleich). Die Regeln unterscheiden sich aber in den erzeugten Platzhaltern für die jeweilige Folgeaktivität (der Kontrollfluss ist also unterschiedlich).



	...	→	…
(3)	I_1	→	$i_1\ R_{I1}$
(4)	$V_1\ R_{I1}$	→	$V_1\ M_1\ P_{X1}$
(5)	$V_1\ R_{I1}$	→	$V_1\ M_2\ P_{X2}$
(6)	$V_1\ R_{I1}$	→	$V_1\ M_3\ P_{X3}$
	...	→	...

Abbildung 27: Unterschiedlicher Kontrollfluss, gleiche Nachrichtentypen

3.2.9.3. Kontrollfluss abhängig von Fehlern

In Kapitel 3.2.2 wurde gezeigt, wie eine Aktivität beendet wird. Dieses Konstrukt beschreibt aber nur, dass sie erfolgreich beendet wird, dass also während der Ausführung dieser Aktivität kein Fehler aufgetreten ist. Eine Aktivität kann aber nicht nur einen Rückgabeparameter, sondern mehrere alternative, sich gegenseitig ausschließende, Rückgabeparameter besitzen. Zum Einen gehört dazu der eigentliche Ergebniswert dieser Aktivität. Zum Anderen kann sie aber auch einen Fehler eines bestimmten Typs zurückgeben.

In WoG ist es möglich frei zu modellieren, was beim Auftreten eines Fehlers passieren soll. Es ist möglich, bedingten Kontrollfluss zu beschreiben, der abhängig vom Auftreten eines Fehlers und vom Typ dieses Fehlers ist. Abbildung 28 zeigt die Behandlung von Fehlern am Beispiel der

Aktivität X_1. Wenn X_1 ohne Fehler beendet wird, dann geht der Kontrollfluss weiter zur Aktivität X_2. Dies bedeutet, dass für alle Kontrollflüsse ohne Fehlerbedingung immer implizit gilt, dass dieser Kontrollfluss nur dann verfolgt wird, wenn kein Fehler auftritt. Tritt bei der Ausführung von X_1 ein Fehler vom Typ F_1 auf, so geht der Kontrollfluss zu X_3. Tritt dagegen ein Fehler vom Typ F_2 auf, so geht der Kontrollfluss zu X_4.

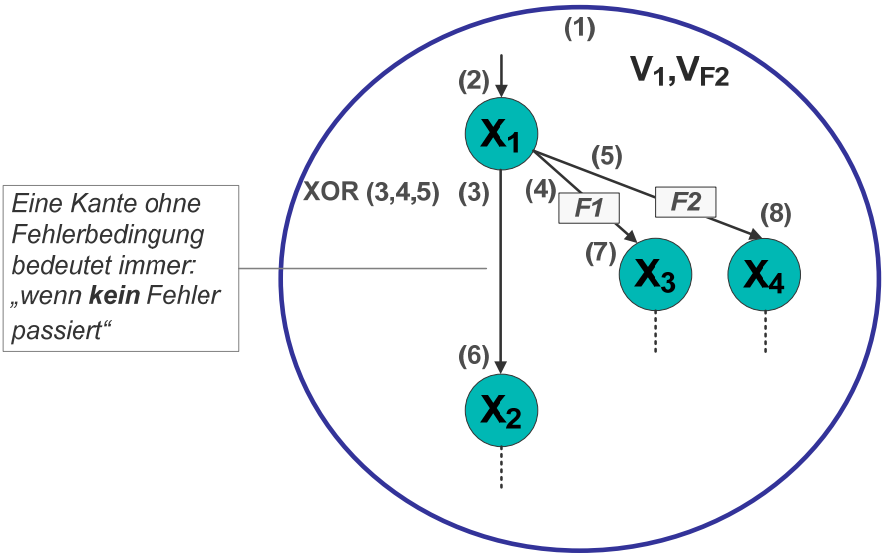


Abbildung 28: Kontrollfluss abhängig von Fehlern

(1)	S	→	$S_{1l}\ V_1\ V_{F2}\ P_{X1}\ S_{1r}$
(2)	P_{X1}	→	X_1
(3)	$V_1\ X_1$	→	$V_1\ x_1\ P_{X2}$
(4)	X_1	→	$x_1\ F_1\ P_{X3}$
(5)	$V_{F2}\ X_1$	→	$V_{F2}\ x_1\ F_2\ P_{X4}$
(6)	P_{X2}	→	X_2
(7)	P_{X3}	→	X_3
(8)	P_{X4}	→	X_4
(9)	F_1	→	ε
(10)	F_2	→	ε
	...	→	...
(I)	$S_{il}\ x_j$	→	$x_j\ S_{il}$
(II)	$V_{i,Sj}\ Y$	→	$Y\ V_{i,Sj}\ , Y \neg\in \{ S_{jl} , S_{jr}\}$
(III)	$Y\ V_{i,Sj}$	→	$V_{i,Sj}\ Y\ , Y \neg\in \{ S_{jl} , S_{jr}\}$

Regelmenge 16: Kontrollfluss abhängig von Fehlern

Regelmenge 16 beschreibt den in Abbildung 28 vorgestellten Prozess. Für die Beendigung der Aktivität X_1 gibt es drei verschiedene Regeln (3), (4) und (5). Regel (3) ist die „normale“ Beendigungsregel, sie wird angewendet wenn kein Fehler auftritt. Regel (4) und (5) behandeln unterschiedliche Fehlertypen. Wenn ein Fehler vom Typ F_1 auftritt, dann wird Regel (4)

angewendet. Tritt ein Fehler vom Typ F_2 auf, dann wird die Regel (5) angewendet. In Regel (5) ist zusätzlich modelliert, dass Fehler vom Typ F_2 bei ihrem Auftreten Fehlerdaten produzieren. Diese Daten werden beim Beenden von X_1 in der Variablen V_{F2} abgelegt.

3.2.10. Explizites Beenden einer laufenden Prozessinstanz

Ein Prozess wird im Normalfall dadurch beendet, dass der Kontrollfluss innerhalb des Prozesses abgearbeitet ist und alle Aktivitäten beendet sind. In manchen Fällen möchte man aber explizit modellieren, dass ein Prozess während seiner Ausführung vorzeitig beendet werden soll. Um dieses Vorgehen in WoG zu beschreiben, wird eine Aktivität „Quit“ (Q_i) sowie spezielle Platzhalter, ein sogenannter Lebensmarker L und ein sogenannter Killmarker K, benötigt. Eine Aktivität vom Typ Quit repräsentiert das vorzeitige und explizite Beenden eines Prozesses. Lebensmarker L und Killmarker K werden zur Realisierung dieses Verhaltens benötigt.

Die Quit Aktivität unterscheidet sich von den bisher vorgestellten Aktivitäten, wie beispielsweise Call oder Evaluation. Diese Aktivitäten repräsentieren bestimmte Aktionen (Aufruf eines Web Services, Auswertung einer Bedingung). Die Ausführung dieser Aktionen wird in WoG aber nicht modelliert (Blackbox). Eine Quit Aktivität repräsentiert das Beenden des gesamten Prozesses. Die Ausführung dieser Aktion wird in WoG durch eine Menge von Regeln explizit modelliert. Ein weiterer Unterschied besteht darin, dass die Aktivierung der Quit Aktivität die Ausführung des gesamten Prozesses beeinflusst.

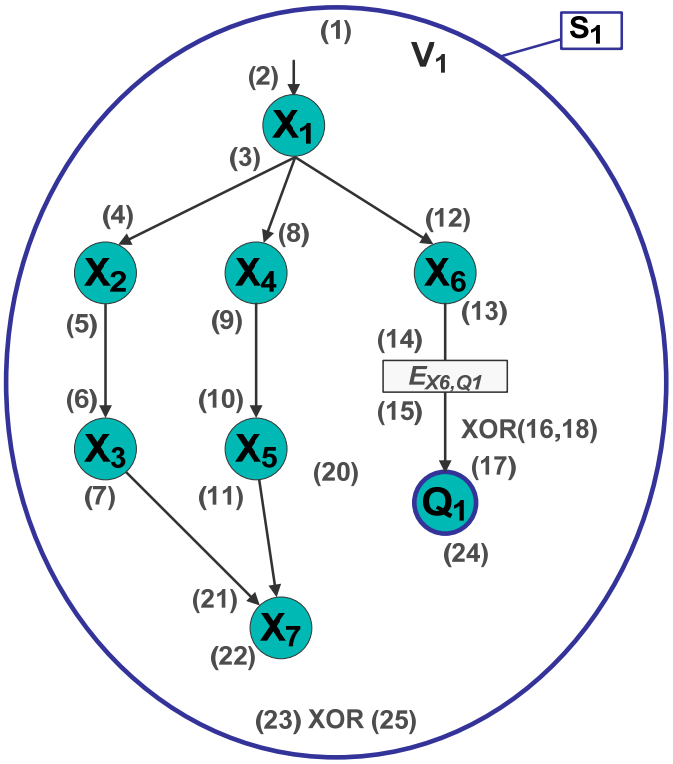


Abbildung 29: Explizites Beenden einer laufenden Prozessinstanz

Abbildung 29 zeigt einen Beispielprozess mit einer Quit Aktivität Q_1. Wenn X_6 beendet wird, wird zunächst die Boolesche Bedingung $E_{X6,Q1}$ ausgewertet. Ist diese „wahr“, so wird die Aktivität Q_1 aktiviert. Diese Aktivität bewirkt, dass alle zu diesem Zeitpunkt noch laufenden Aktivitäten im gesamten Prozess beendet werden und anschließend der Prozess selbst ebenfalls beendet wird. Das bedeutet insbesondere, dass Aktivitäten während ihrer Ausführung vorzeitig beendet werden.

Die Modellierung dieses Beispielprozesses in WoG wird in der Regelmenge 17 (Prozessstrukturregeln) und Regelmenge 18 (generische Regeln) dargestellt.

(1)	S	→	$S_{1l}\ K_l\ L\ V_1\ P_{X1}\ K_r\ S_{1r}$
(2)	$L\ P_{X1}$	→	$L\ X_1$
(3)	$L\ X_1$	→	$L\ x_1\ P_{X2}\ P_{X4}\ P_{X6}$
(4)	$L\ P_{X2}$	→	$L\ X_2$
(5)	$L\ X_2$	→	$L\ x_2\ P_{X3}$
(6)	$L\ P_{X3}$	→	$L\ X_3$
(7)	$L\ X_3$	→	$L\ x_3\ P_{X7m}$
(8)	$L\ P_{X4}$	→	$L\ X_4$
(9)	$L\ X_4$	→	$L\ x_4\ P_{X5}$
(10)	$L\ P_{X5}$	→	$L\ X_5$
(11)	$L\ X_5$	→	$L\ x_5\ P_{X7}$
(12)	$L\ P_{X6}$	→	$L\ X_6$
(13)	$L\ X_6$	→	$L\ x_6\ P_{E\_X6,Q1}$
(14)	$L\ V_1\ P_{E\_X6,Q1}$	→	$L\ V_1\ E_{X6,Q1}$
(15)	$L\ E_{X6,Q1}$	→	$L\ R_{E\_X6,Q1}$
(16)	$L\ R_{E\_X6,Q1}$	→	$L\ T\ P_{Q1}$
(17)	$L\ P_{Q1}$	→	$Q_1\ K$
(18)	$L\ R_{E\_X6,Q1}$	→	$L\ F$
(19)	$L\ T$	→	L
(20)	$L\ F$	→	L
(21)	$L\ P_{X7m}\ P_{X7}$	→	$L\ X_7$
(22)	$L\ X_7$	→	$L\ x_7$
(23)	$S_{1l}\ K_l\ L\ V_1\ K_r\ S_{1r}$	→	ε
(24)	$K_l\ Q_1\ K_r$	→	q_1
(25)	$S_{1l}\ K\ S_{1r}$	→	ε

Regelmenge 17: Prozess mit Quit – Prozessstrukturregeln

(I)	$S_{il}\ x_j$	→	$x_j\ S_{il}$	
(II)	$V_{i,Sj}\ Y$	→	$Y\ V_{i,Sj}$	, $Y \neg\in \{ S_{jl} , S_{jr}\}$
(III)	$Y\ V_{i,Sj}$	→	$V_{i,Sj}\ Y$	, $Y \neg\in \{ S_{jl} , S_{jr}\}$
(IV)	$P_{Xim}\ Y$	→	$Y\ P_{Xim}$	, $Y \in NT \cup T$
(V)	$Y\ P_{Xim}$	→	$P_{Xim}\ Y$	, $Y \in NT \cup T$
(VI)	$L\ Y$	→	$Y\ L$	, $Y \in NT \cup T$
(VII)	$Y\ L$	→	$L\ Y$	, $Y \in NT \cup T$
(VIII)	$K\ Y$	→	$Y\ K$	, $Y \in NT \cup T$
(IX)	$Y\ K$	→	$K\ Y$	, $Y \in NT \cup T$
(X)	$K\ X_j$	→	$K\ x_j$	, $X_j != Q_i$
(XI)	$K\ P_j$	→	K	
(XII)	$K\ V_j$	→	K	
(XIII)	$K\ R_j$	→	K	
(XIV)	$K\ T$	→	K	
(XV)	$K\ F$	→	K	
(XVI)	$K\ M_j$	→	K	
(XVII)	$K\ F_j$	→	K	
(XVIII)	$K\ S_{il}$	→	K	, i != 1 (Prozessscope)
(XIX)	$K\ S_{ir}$	→	K	, i != 1 (Prozessscope)
(XX)	$K_l\ x_i$	→	$x_i\ K_l$	
(XXI)	$x_i\ K_r$	→	$K_r\ x_i$	

Regelmenge 18: Prozess mit Quit – Generische Regeln

Beim Starten des Prozesses wird in Regel (1) der Prozessscope mit der Variable V_1 und dem Platzhalter P_{X1} erzeugt. Zusätzlich werden Platzhalter K_l, K_r und L erzeugt. Die Bedeutung und Verwendung von K_l und K_r wird später erklärt. Der Platzhalter L (Lebensmarker) signalisiert, dass der Prozess normal ausgeführt wird. Dieser Lebensmarker kommt in allen Regeln vor, die den normalen Kontrollfluss und nicht die vorzeitige Beendigung des Prozesses modellieren. Das bedeutet, dass diese Regeln nur angewendet werden können, solange der Lebensmarker L existiert. Die generischen Regeln (VI) und (VII) beschreiben, dass der Lebensmarker frei beweglich ist.

Der Lebensmarker kann nur durch die Aktivierung einer Quit Aktivität gelöscht werden. Damit wird erreicht, dass die Aktivierung einer Quit Aktivität Auswirkungen auf den gesamten Prozess hat. Sobald eine Quit Aktivität aktiviert worden ist, können die „normalen“ Prozessstrukturregeln nicht mehr angewendet werden.

In Regel (17) wird die Aktivität Q_1 aktiviert, der Lebensmarker L gelöscht, sowie das Nichtterminal K (Killmarker) erzeugt. Dieser Killmarker bewirkt, dass die Regeln, welche das Beenden des gesamten Prozesses modellieren (VIII bis XXI sowie 24 und 25), angewendet werden können.

Die Regeln (VIII) und (IX) besagen, dass der Killmarker sich im gesamten Wort frei bewegen kann. Regel (X) sagt aus, das der Killmarker alle aktiven Aktivitäten, ausgenommen Q_1, beendet. Das bedeutet insbesondere, dass Aktivitäten während ihrer Ausführung vorzeitig beendet werden. Die

Regeln (XI) bis (XIX) beschreiben, dass alle weiteren Nichtterminale, ausgenommen der Grenzen des Prozessscopes, gelöscht werden.

Die Nichtterminale K_l und K_r werden benötigt, um festzustellen, wann die Aktivität Q_1 selbst beendet werden kann. Die Regeln (XX) und (XXI) sagen aus, das die linke Grenze des Killmarkers (K_l) nur nach rechts und die rechte Grenze des Killmarkers (K_r) nur nach links wandern kann. Beide Marker können dabei ausschließlich über beendete Aktivitäten wandern. Regel (24) beendet die Quit Aktivität. Die Killmarkergrenzen stellen sicher, dass diese Regel erst dann angewendet werden kann, wenn alle Aktivitäten (mit Ausnahme von Q_1) beendet worden sind und alle weiteren Nichtterminale (mit Ausnahme der Prozessscopegrenzen) gelöscht worden sind. In Regel (25) wird schließlich der Prozess beendet.

3.3. Erweiterte WoG Konstrukte

In Kapitel 3.1 und 3.2 wurden die Aktivitäten und Modellierungskonstrukte von WoG vorgestellt. Im Folgenden wird gezeigt, wie diese kombiniert werden können, um erweiterte Strukturen zu modellieren. Es werden verschiedene Programmierkonstrukte betrachtet, wie sie aus vielen Programmiersprachen bekannt sind [KS09]. Der am Ende des vorherigen Kapitels eingeführte Lebensmarker wird aus Gründen der Übersichtlichkeit in den folgenden Regelmengen nicht dargestellt.

3.3.1. if then else

“if then else” ist ein klassisches Programmierkonstrukt. Im Folgenden wird gezeigt, wie einzelne Varianten von „if then else“ mit den bisher vorgestellten WoG Konstrukten modelliert werden können.

3.3.1.1. if (ohne else Zweig)

In Kapitel 3.2.9.1 wurde beschrieben, wie Kontrollfluss abhängig von einer Booleschen Bedingung in WoG umgesetzt werden kann. Der dort beschriebene bedingte Kontrollfluss entspricht einer „if“ Anweisung ohne „else“ Zweig.

3.3.1.2. if then else

„if then else“ beschreibt einen alternativen, sich gegenseitig ausschließenden Kontrollfluss. Ein „if then else“ modelliert dabei genau zwei Alternativen.

In Abbildung 24 wird ein Beispielprozess mit einem „if then else“ Konstrukt auf der linken Seite grafisch und auf der rechten Seite mit Pseudocode dargestellt. Nachdem die Aktivität X_1 beendet worden ist, wird im nächsten Schritt die Boolesche Bedingung $E_{X2,X3}$ ausgewertet. Wenn das Ergebnis dieser Auswertung der Wahrheitswert „wahr“ ist, dann wird die Aktivität X_2 aktiviert. Wenn das Ergebnis dieser Auswertung der Wahrheitswert „falsch“ ist, dann wird die Aktivität X_3 aktiviert.

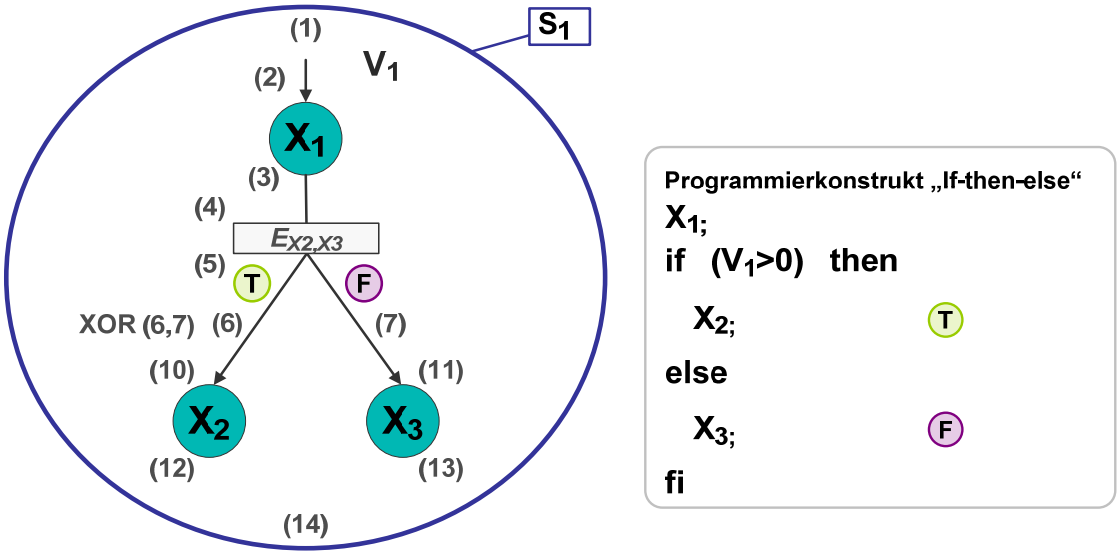


Abbildung 30: if then else

In WoG wird „if then else“ mit Hilfe des in Kapitel 3.2.9.1 vorgestellten WoG Konstruktes „Kontrollfluss abhängig von Booleschen Bedingungen“ modelliert. Der in Abbildung 30 vorgestellte Beispielprozess wird in Regelmenge 19 beschrieben. In Regel (4) und (5) wird die Auswertung der booleschen Bedingung beschrieben. Der alternative Kontrollfluss wird durch die Regeln (6) und (7) realisiert. Abhängig vom Ergebnis der Auswertung der Bedingung wird ein entsprechender Platzhalter erzeugt. Wird die Bedingung zu „wahr“ ausgewertet, so wird Regel (6) angewendet und der Platzhalter P_{X2} für die Aktivität X_2 erzeugt („if“ Zweig). Wird die Bedingung zu „falsch“ ausgewertet, so wird Regel (7) angewendet und der Platzhalter P_{X3} für die Aktivität X_3 erzeugt („else“ Zweig).

(1)	S	→	$S_{1l}\ V_1\ P_{X1}\ S_{1r}$	
(2)	P_{X1}	→	X_1	
(3)	X_1	→	$x_1\ P_{E\_X2,X3}$	
(4)	$V_1\ P_{E\_X2,X3}$	→	$V_1\ E_{X2,X3}$	
(5)	$E_{X2,X3}$	→	$e_{X2,X3}\ R_{E\_X2,X3}$	
(6)	$R_{E\_X2,X3}$	→	$T\ P_{X2}$	T
(7)	$R_{E\_X2,X3}$	→	$F\ P_{X3}$	F
(8)	T	→	ε	
(9)	F	→	ε	
(10)	P_{X2}	→	X_2	
(11)	P_{X3}	→	X_3	
	...	→	...	

Regelmenge 19: if then else

3.3.1.3. if then else innerhalb einer Sequenz

Abbildung 31 zeigt die Verwendung eines „if then else“ Konstruktes innerhalb einer Sequenz. Dies bedeutet insbesondere, dass der Kontrollfluss sowohl nach der Abarbeitung des „if“ Zweiges, als auch nach der Abarbeitung des „else“ Zweiges an der gleichen Stelle weitergeht. Im dargestellten Beispielprozess ist dies die Aktivität X_4.

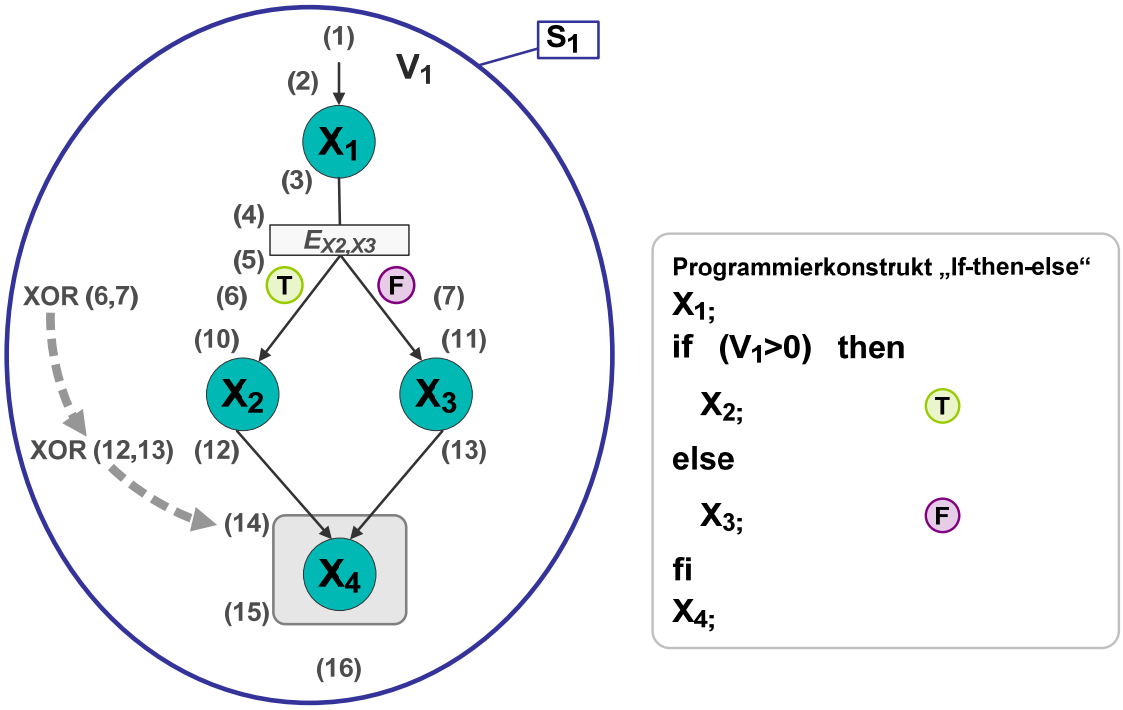
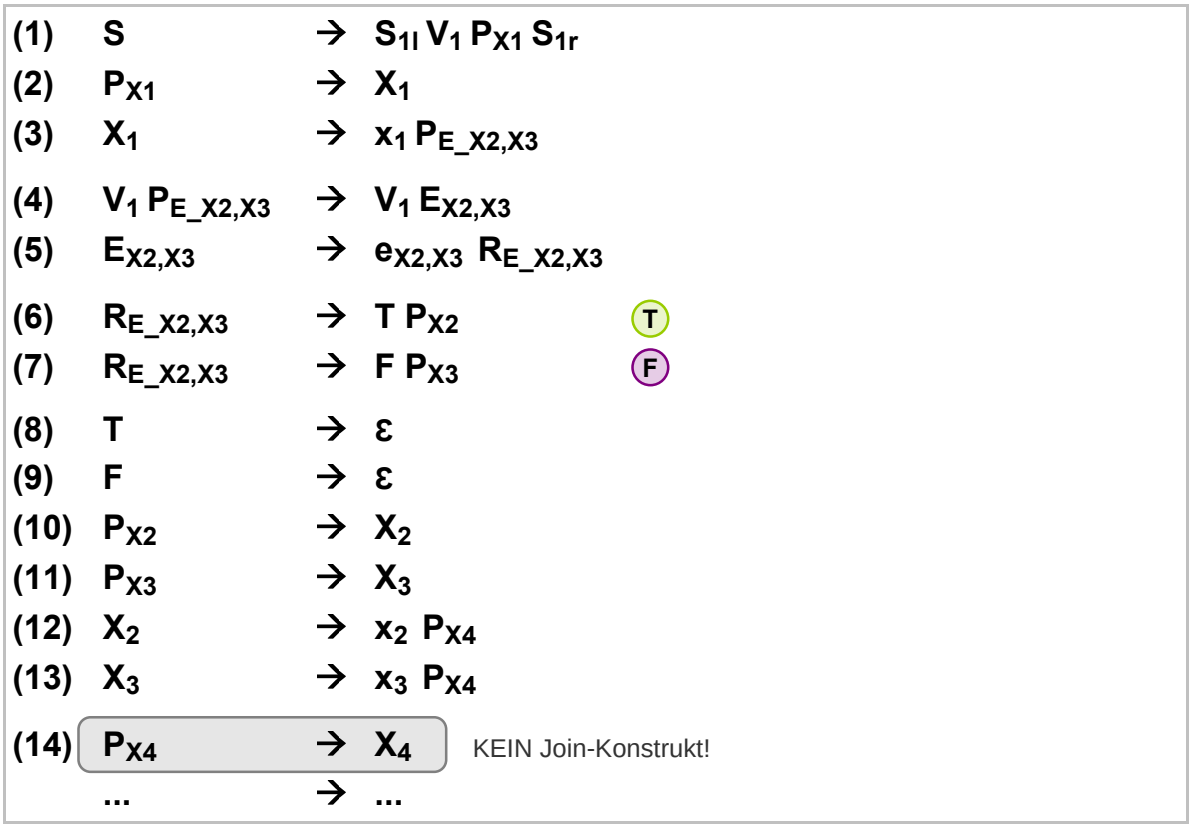


Abbildung 31: if then else innerhalb einer Sequenz

Der in Abbildung 31 dargestellte Prozess wird in WoG mit der Regelmenge 20 modelliert. Jeder der beiden alternativen Pfade wird so aufgebaut, als ob es den anderen Zweig nicht geben würde. Dies entspricht genau der Bedeutung sich gegenseitig ausschließender Alternativen. Die Regeln (6), (10), (12) und (14) modellieren eine ganz normale Sequenz von X_2 und X_4. Die Regeln (7), (11), (13) und (14) tun das Gleiche für die Sequenz von X_3 und X_4. Durch die Regeln (12) und (13) gibt es zwei Möglichkeiten, die Aktivität X_4 zu aktivieren. Zur Laufzeit wird aber genau ein Platzhalter P_{X4} erzeugt. Die Regel (12) wird nur beim Durchlauf des „if“ Zweiges angewendet, die Regel (13) nur beim Durchlauf des „else“ Zweiges. Da sich diese Zweige gegenseitig ausschließen, wird genau eine der beiden Regeln angewendet, X_4 wird also genau einmal aktiviert.



Regelmenge 20: if then else innerhalb einer Sequenz

3.3.1.4. if then elseif

„if then elseif“ beschreibt, wie „if then else“ auch, einen alternativen, sich gegenseitig ausschließenden Kontrollfluss. Mit „if then elseif“ kann man aber beliebig viele sich gegenseitig ausschließende Alternativen modellieren. Ein „if then elseif“ Konstrukt kann immer auch durch mehrere verschachtelte „if then else“ Konstrukte modelliert werden. Abbildung 32 zeigt eine solche Umwandlung an einem Beispiel. In WoG wird ein „if then elseif“ Konstrukt als verschachteltes „if then else“ modelliert.

Das in Abbildung 32 abgebildete Beispiel wird in Abbildung 33 grafisch dargestellt. Der Prozess enthält eine Reihe von Auswertungsaktivitäten (E_i). Diese werten einen booleschen Ausdruck aus. Wird dieser als wahr ausgewertet, wird eine Aktivität X_j aktiviert. Andernfalls wird die Auswertung des nächsten Ausdrucks aktiviert. Werden alle Auswertungen zu „false“ ausgewertet“, dann wird die Aktivität X_5 aktiviert. In allen Fällen geht der Kontrollfluss anschließend bei X_6 weiter.

```
// if then elseif
X1;
if  (V1 > 50)  then
  X2;
elseif  (V1 > 30)  then
  X3;
elseif  (V1 > 10)  then
  X4;
else
  X5;
fi
X6;
```

```
// verschachteltes if then else
X1;
if  (V1 > 50)  then
  X2;
else
  if  (V1 > 30)  then
    X3;
  else
    if  (V1 > 10)  then
      X4;
    else
      X5;
    fi
  fi
fi
X6;
```

Abbildung 32: if then elseif – Pseudocode

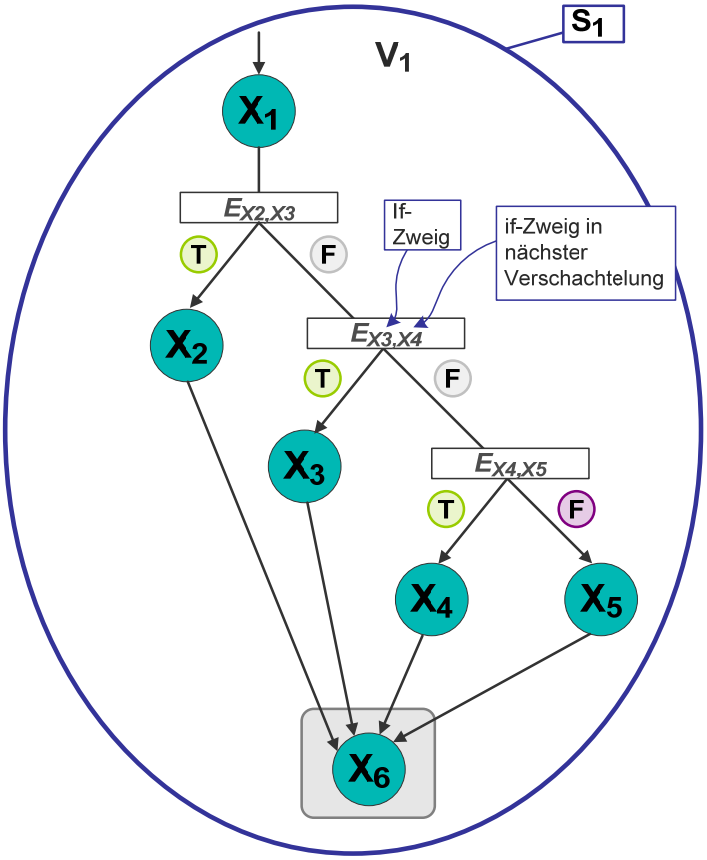
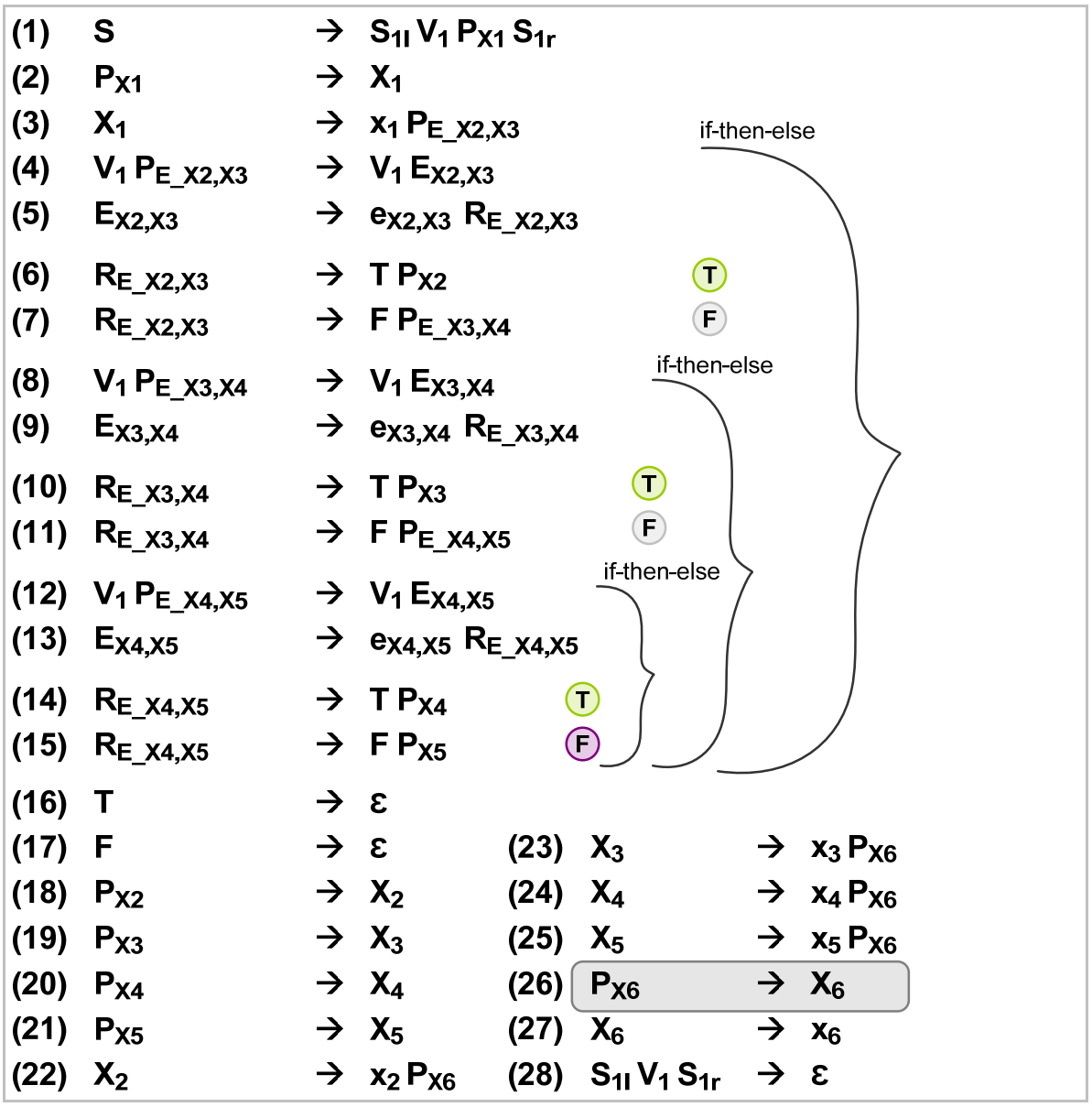


Abbildung 33: if then elseif

Regelmenge 21 modelliert diesen Prozess in WoG. Regel (7) und (11) stellen die Verschachtelung von „if then else“ Konstrukten dar. Wird eine Bedingung zu falsch ausgewertet, aktivieren diese

Regeln die Auswertung der Bedingung des nächsten „if then else“ Konstruktes. Ähnlich wie bei der Modellierung eines „if then else innerhalb einer Sequenz“ gilt hier, dass zur Laufzeit genau eine der Regeln (22) bis (25) angewendet wird. Die Aktivität X_6 wird also genau einmal aktiviert.



Regelmenge 21: if then elseif

3.3.2. While (Schleifenkonstrukt)

Eine „while“ Schleife ist die allgemeine Form der Schleife. Sie besteht aus einer Schleifenbedingung und einem Schleifenkörper. Wenn die Schleifenbedingung war ist, dann wird der Schleifenkörper genau einmal ausgeführt. Dies wird solange wiederholt, bis die Schleifenbedingung zum ersten Mal zu „falsch“ ausgewertet wird. Andere Schleifentypen, wie zum Beispiel „repeat until“ oder „for“ Schleifen können durch eine „while“ Schleife dargestellt werden.

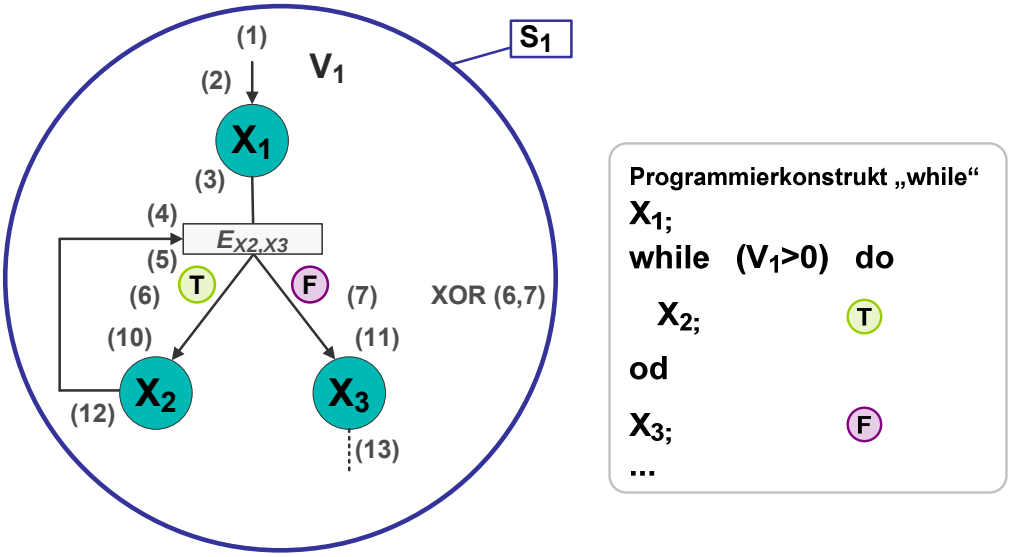


Abbildung 34: while Schleife

In Abbildung 34 wird ein Beispielprozess mit einer „while“ Schleife auf der linken Seite grafisch und auf der rechten Seite mit Pseudocode dargestellt. Die Schleife beginnt nach Beendigung der Aktivität X_1. Die Boolesche Bedingung $E_{X2,X3}$ ist die Schleifenbedingung, die Aktivität X_2 ist der Schleifenkörper. Nach dem Ende der Schleife wird die Aktivität X_3 ausgeführt.

(1)	S	→	$S_{1l}\ V_1\ P_{X1}\ S_{1r}$	
(2)	P_{X1}	→	X_1	
(3)	X_1	→	$x_1\ P_{E\_X2,X3}$	
(4)	$V_1\ P_{E\_X2,X3}$	→	$V_1\ E_{X2,X3}$	
(5)	$E_{X2,X3}$	→	$e_{X2,X3}\ R_{E\_X2,X3}$	
(6)	$R_{E\_X2,X3}$	→	$T\ P_{X2}$	T
(7)	$R_{E\_X2,X3}$	→	$F\ P_{X3}$	F
(8)	T	→	ε	
(9)	F	→	ε	
(10)	P_{X2}	→	X_2	
(11)	P_{X3}	→	X_3	
(12)	X_2	→	$x_2\ P_{E\_X2,X3}$	
(13)	X_3	→	x_3 ...	
	...	→	...	

Regelmenge 22: while Schleife

In WoG wird eine „while“ Schleife mit Hilfe des in Kapitel 3.2.9.1 vorgestellten WoG Konstruktes „Kontrollfluss abhängig von Booleschen Bedingungen“ modelliert. Der in Abbildung 34 vorgestellte Beispielprozess wird in Regelmenge 22 beschrieben. In Regel (4) und (5) wird die Auswertung der Schleifenbedingung beschrieben. Abhängig vom Ergebnis wird entweder mit Regel (6) der Schleifenkörper aktiviert oder mit Regel (7) die Schleife beendet und ihre Folgeaktivität X_3 aktiviert.

Nach einem Durchlauf des Schleifenkörpers wird in Regel (12) erneut der Platzhalter $P_{E\_X2,X3}$ für die Schleifenbedingung erzeugt.

3.4. Zusammenfassung

Am Anfang von Kapitel 3 werden die Grundlagen der Workflow Modellierungssprache WoG beschrieben. Eine WoG Grammatik beschreibt ein Prozessmodell durch ihre Produktionsregeln. Eine Produktionsregel entspricht einem einzelnen Schritt in der Prozessausführung. Eine Ableitung eines Wortes mit einer WoG Grammatik entspricht einer Ausführung des durch die Grammatik beschriebenen Prozessmodells. Jedes Teilwort der Ableitung beschreibt einen Prozesszustand während dieser Ausführung.

In WoG wird Kontrollfluss mit Hilfe der Produktionsregeln explizit modelliert. Die Beendigungs- und Aktivierungsregeln einzelner Aktivitäten werden durch Platzhalter miteinander verknüpft. Daten werden in WoG durch Variablen repräsentiert. Durch die Verwendung von Variablen in Aktivierungs- und Beendigungsregeln von Aktivitäten wird explizit dargestellt, welche Aktivität auf welche Variablen lesend oder schreibend zugreift. Datenfluss wird in WoG implizit modelliert. Die Modellierung von Datenfluss ohne entsprechenden Kontrollfluss ist in WoG möglich.

Id	**Name**	**Beschreibung**
C_i	Call	Die Call Aktivität ruft einen Web Service auf.
I_i	In	Die In Aktivität kann von außen aufgerufen werden und auf diese Weise Nachrichten empfangen.
O_i	Out	Die Out Aktivität gehört immer zu einer In Aktivität. Nach dem Empfangen einer Nachricht kann mit der Out Aktivität eine dazugehörende Antwortnachricht zurückgesendet werden.
A_i	Assign	Mit der Assign Aktivität modelliert man in WoG eine oder mehrere Zuweisungen.
E_i	Evaluation	Die Aktivität Evaluation wertet einen booleschen Ausdruck aus.
W_i	Wait	Mit der Wait Aktivität kann „warten“ modelliert werden.
Q_i	Quit	Eine Aktivität vom Typ Quit repräsentiert das vorzeitige und explizite Beenden eines Prozesses.

Tabelle 2: Aktivitätstypen in WoG

In Kapitel 3.1 werden die WoG Aktivitätstypen vorgestellt. Diese sind in Tabelle 2 noch einmal gesammelt dargestellt und kurz beschrieben. In Kapitel 3.2 werden die Modellierungskonstrukte von WoG spezifiziert. Tabelle 3 zeigt alle Nichtterminaltypen (außer den Aktivitätstypen), die in diesen Konstrukten verwendet werden. In Kapitel 3.3 wird gezeigt, wie die vorgestellten Modellierungskonstrukte von WoG verwendet werden können, um erweiterte Konstrukte zu modellieren.

Id	**Name**	**Beschreibung**
V_i	Variable	Eine Variable i wird durch genau ein Nichtterminal V_i repräsentiert
R_i	Temporäre Ergebnisvariable	R_i ist eine spezielle temporäre Variable, deren Inhalt zur Laufzeit eines Prozesses gelesen und interpretiert wird.
P_i, P_{im}	Platzhalter (fest und mobil)	P_i ist ein Nichtterminal ohne weitere Semantik. Mobile Platzhalter P_{im} können im Wort frei bewegt werden.

T	True	Das Nichtterminal T repräsentiert den Wahrheitswert „wahr“ („true“).
F	False	Das Nichtterminal F repräsentiert den Wahrheitswert „falsch“ („false“).
L	Lebensmarker	Der Lebensmarker signalisiert, dass der Prozess normal ausgeführt wird.
K	Killmarker	Der Killmarker wird verwendet, um einen Prozess vorzeitig zu beenden.
K_l, K_r	Grenzen des Killmarkers	Die Nichtterminale K_l und K_r stellen das korrekte Verhalten des Killmarkers sicher.
M_i	MessageInterface	Das Nichtterminal M_i repräsentiert den Empfang einer Nachricht über eine bestimmte Schnittstelle.
F_i	Fehlertyp	Das Nichtterminal F_i repräsentiert einen Fehler eines bestimmten Typs.
S_{il}, S_{ir}	Scopegrenzen	Jeder Scope S_i wird durch eine linke und eine rechte Scopegrenze repräsentiert.
S	Startsymbol	Jede Grammatik besitzt genau ein Startsymbol.

Tabelle 3: Nichtterminaltypen in WoG

4 Dateiformat für WoG

In diesem Kapitel wird ein XML basiertes Dateiformat für WoG Grammatiken definiert. In Kapitel 4.1 wird zunächst die in dieser Arbeit gewählte Darstellungsform beschrieben. Anschließend wird in Kapitel 4.2 die Struktur des WoG XML Dateiformats vorgestellt.

4.1. Darstellung der XML Struktur

Sowohl BPEL als auch WoG Prozessmodelle werden in XML dargestellt. Um die XML Struktur einzelner Elemente zu beschreiben, wird in dieser Arbeit nicht XML Schema, sondern eine vereinfachte Darstellung verwendet. Diese Darstellungsweise wird anhand des in Listing 1 gezeigten Beispiels vorgestellt. In diesem Beispiel wird der Aufbau eines „receive“ Elementes beschrieben. Ein solches Element besitzt ein Pflichtattribut mit dem Namen „partnerLink“ und dem Datentyp „NCName“ (Zeile 1). In Zeile 2 ist das optionale Attribut „portType“ dargestellt. Das Fragezeichen „?“ am Ende der Zeile symbolisiert, dass dieses Attribut einmal vorkommen kann, aber nicht vorkommen muss. Dementsprechend wird in Zeile 5 dargestellt, dass ein „receive“ Element ein optionales Kindelement „correlations“ haben kann. Dieses Element hat wiederum ein oder mehrere Kindelemente „correlation“. Das Pluszeichen „+“ am Ende von Zeile 7 bedeutet, dass ein Element mindestens einmal vorkommen muss und mehr als einmal vorkommen darf. Das in diesem Beispiel nicht gezeigte Zeichen „*“ symbolisiert, dass ein Element beliebig oft vorkommen kann, aber nicht vorkommen muss. In Zeile 7 wird im Attribut „initiate“ explizit angegeben, welche Werte dieses Attribut annehmen kann. Das Zeichen „|“ beschreibt eine Alternative. Die Zeichenfolge „…“ bedeutet, dass nicht alle Elemente oder Attribute dargestellt werden.

```
[01] <receive partnerLink="NCName"
[02]          portType="QName"?
[03]          … >
[04]    …
[05]   <correlations>?
[06]     <correlation set="NCName"
[07]                  initiate="yes|join|no"? />+
[08]   </correlations>
[09]   …
[10] </receive>
```

Listing 1: Beispiel für die Darstellung einer XML Struktur

4.2. Aufbau des WoG XML Dateiformats

Die Struktur des hier vorgestellten Dateiformats für WoG orientiert sich an der Definition von Grammatiken. Eine Grammatik besteht aus einer Menge von Nichtterminalen, einer Menge von Terminalen, einer Menge von Produktionsregeln und einem Startsymbol. Listing 2 zeigt den Aufbau des Dateiformats für WoG Grammatiken. Es wird im Folgenden näher beschrieben.

Die Menge der Nichtterminale ist in den Zeilen 3 bis 10 dargestellt. Sie muss mindestens ein Nichtterminal enthalten, da eine Grammatik immer das Startsymbol enthalten muss. In den Zeilen 4 bis 9 ist der allgemeine Aufbau eines Nichtterminals dargestellt. Jedes Nichtterminal besitzt eine eindeutige Id (Zeile 4). Diese wird benötigt, um ein Nichtterminal referenzieren zu können. Die Benennung eines Nichtterminals ist optional (Zeile 5). Nichtterminale können von unterschiedlichen

Typen sein (Zeile 6). Abhängig von seinem Typ kann ein Nichtterminal noch weitere Attribute oder Elemente enthalten. Die einzelnen Typen werden in Kapitel 3 eingeführt, ihre Attribute und Elemente werden in Kapitel 5 detailliert beschrieben.

Die Menge der Terminale ist in Zeile 12 bis 16 beschrieben. Sie kann beliebig viele Terminale enthalten, aber auch leer sein. In WoG repräsentieren Terminale immer beendete Aktivitäten, laufende Aktivitäten werden als Nichtterminale repräsentiert. In Zeile 14 ist dargestellt, dass ein Terminal lediglich eine Referenz auf ein Nichtterminal ist. Von Terminalen referenzierte Nichtterminale müssen Aktivitäten sein. Das Attribut „target“ gibt die Id (Attribut „id“) des referenzierten Nichtterminals an.

```
[01] <wogGrammar xmlns="http://iaas.uni-stuttgart.de/wog/model"
[02]             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
[03]   <nonTerminalSet>
[04]     <nonTerminal id="NCName"
[05]                  name="NCName"?
[06]                  xsi:type="tCall|tOut|…|tPlaceholder"
[07]                  … >+
[08]       …
[09]     </nonTerminal>
[10]   </nonTerminalSet>
[11]
[12]   <terminalSet>
[13]     <terminal id="NCName">*
[14]       <nonTerminalReference target="NCName" />
[15]     </terminal>
[16]   </terminalSet>
[17]
[18]   <ruleSet>
[19]     <rule name="NCName"? >*
[20]       <ruleLeftSide>
[21]         (
[22]         <nonTerminalReference target="NCName" />
[23]         |
[24]         <terminalReference target="NCName" />
[25]         )+
[26]       </ruleLeftSide>
[27]       <ruleRightSide>
[28]         (
[29]         <nonTerminalReference target="NCName" />
[30]         |
[31]         <terminalReference target="NCName" />
[32]         )*
[33]       </ruleRightSide>
[34]     </rule>
[35]   </ruleSet>
[36]
[37]   <startSymbol>
[38]     <nonTerminalReference target="NCName" />
[39]   </startSymbol>
[40] </wogGrammar>
```

Listing 2: Dateiformat für WoG, XML Struktur

Die Regelmenge einer WoG Grammatik wird in den Zeilen 18 bis 35 dargestellt. Sie kann beliebig viele Regeln beinhalten. Eine Regel ist unterteilt in eine linke und eine rechte Seite. Der Aufbau der linken Seite ist in Zeile 20 bis 26 dargestellt. Die linke Seite einer Regel besteht aus beliebig vielen Terminalen oder Nichtterminalen, sie muss jedoch mindestens ein Nichtterminal enthalten. Terminale und Nichtterminale werden hier als Referenzen auf Elemente der Nichtterminal- oder

Terminalmenge angegeben. Die rechte Seite einer Regel (Zeile 27 bis 33) ist ähnlich aufgebaut wie die linke Seite. Der einzige Unterschied besteht darin, dass sie keine Elemente enthalten muss.

Das Startsymbol der Grammatik wird in den Zeilen 37 bis 39 angegeben. Es besteht aus einer Referenz auf ein Nichtterminal. Das referenzierte Nichtterminal muss vom Typ „tStartSymbol“ sein.

5 Transformation von BPEL nach WoG

In Kapitel 3 wurde die Workflow Sprache WoG vorgestellt. WoG wurde mit dem Ziel entworfen, dass verschiedenste Workflow Sprachen auf WoG abbildbar sind. In diesem Kapitel wird gezeigt, wie BPEL Prozessmodelle auf WoG Grammatiken abgebildet werden können. BPEL ist eine komplexe und sehr mächtige Workflow Sprache [JMS06]. Durch ihre Abbildung auf WoG soll gezeigt werden, dass WoG ebenfalls die Modellierung sehr komplexer und mächtiger Prozessmodelle erlaubt.

In Kapitel 5.1 werden zunächst Konventionen festegelegt und einige allgemeine Aspekte der Transformation beschrieben. Kapitel 5.2 zeigt, dass die Transformation von BPEL nach WoG modular strukturiert werden kann. Durch diese Strukturierung ist es möglich, die Übersetzung einzelner Aktivitäten losgelöst vom restlichen Prozess zu definieren. Kapitel 5.3 betrachtet das „process“ Element des BPEL Prozesses und beschreibt seine Abbildung auf WoG. Damit wird der Rahmen festgelegt, in dem die weitere Abbildung der einzelnen Aktivitäten eines BPEL Prozesses erfolgen kann. Kapitel 5.4 beschreibt die Abbildung von BPEL Basisaktivitäten auf WoG. Kapitel 5.5 behandelt strukturierte BPEL Aktivitäten. In Kapitel 5.6 wird die Transformation von BPEL nach WoG (BPEL2WoG) noch einmal zusammengefasst.

Die hier vorgestellte Transformation von BPEL nach WoG umfasst einen Großteil der in [BPEL07] definierten BPEL Konstrukte. „Cross Boundary Links“ (Kapitel 5.5.6) und „Isolated Scopes“ (Kapitel 5.5.8) werden in dieser Arbeit nicht betrachtet.

5.1. Allgemeines

5.1.1. Konventionen

XML Struktur

Sowohl BPEL als auch WoG Prozessmodelle werden in XML dargestellt. Um die XML Struktur einzelner Elemente zu beschreiben, wird in dieser Arbeit nicht XML Schema, sondern eine vereinfachte Darstellung verwendet. Diese wird in Kapitel 4.1 eingeführt und auch in Kapitel 5 verwendet.

WoG Regelmengen

Bei der Darstellung von WoG Regelmengen werden die Beginn- und Endplatzhalter der jeweils beschriebenen BPEL Aktivität umrandet hervorgehoben. Die Bedeutung dieser Platzhalter wird in Kapitel 5.2 erklärt. Der Beginnplatzhalter einer BPEL Aktivität Y wird mit P_{Yb} bezeichnet, der Endeplatzhalter mit P_{Ye}.

Der in Kapitel 3.2.10 „Explizites Beenden einer laufenden Prozessinstanz“ eingeführte Lebensmarker wird aus Gründen der Übersichtlichkeit in den Regelmengen nicht dargestellt. Lediglich in den Beispielen, für die er wesentlich ist, ist er in den Regeln enthalten. Gleiches gilt für die generischen Regeln. Sie werden nur dort dargestellt, wo sie notwendig sind.

5.1.2. Datenmodell und Web Service Kommunikation

Das primäre Datenformat von BPEL basiert auf XML Schema. Der Typ einer Variablen wird durch XML Schema definiert. Ausdrücke werden als XPath angegeben. Gleiches gilt für WoG. In den bisherigen Kapiteln wurden WoG Variablen abstrakt als Datencontainer betrachtet. In diesem Kapitel werden ihr Aufbau und ihre Verwendung näher ausspezifiziert.

Kommunikation erfolgt in BPEL Prozessen über Web Service Schnittstellen. BPEL wird auch als rekursive Aggregationssprache für Web Services bezeichnet. BPEL Prozesse rufen Web Services auf und sind selbst als Web Service aufrufbar. BPEL beinhaltet ein ausführliches Konzept zur Beschreibung komplexer Kommunikationsszenarien. Die Spezifikation und das Zusammenspiel dieser Mechanismen wird im BPEL Standard ausführlich erklärt [BPEL07]. Dieses Konzept wird für WoG übernommen und im Folgenden kurz beschrieben.

Ein „PartnerLink“ beschreibt eine Interaktion zwischen zwei Teilnehmern. Zur Laufzeit werden die Endpunktreferenzen, also die konkreten Adressen der Teilnehmer, im PartnerLink hinterlegt.

„Properties“ und „PropertyAlias“ definieren globale Datenelemente, die für die Prozesslogik wichtig sind. Diese können verwendet werden, um den Zugriff auf Variablen zu vereinfachen und zu vereinheitlichen. Sie werden ebenfalls verwendet, um Nachrichten und Prozessinstanzen einander zuordnen zu können. Dies geschieht mit der Hilfe von „CorrelationSets“. Ein CorrelationSet gibt an, welche Properties eine Prozessinstanz eindeutig identifizieren. Eine Nachricht und eine Prozessinstanz gehören zusammen, wenn ihre im CorrelationSet angegebenen Properties den gleichen Wert haben.

Ein BPEL Prozess kann Nachrichten empfangen, verarbeiten und anschließend eine Antwortnachricht senden. Wenn es mehrere empfangende und sendende Aktivitäten in einem Prozess gibt, dann kann „messageExchange“ dazu verwendet werden, diese Aktivitäten eindeutig einander zuzuordnen.

5.1.3. Allgemeine Attribute und Elemente von BPEL Aktivitäten

BPEL definiert sowohl Attribute als auch Elemente, die in jeder Aktivität enthalten sein können. Diese Attribute und Elemente sind in Listing 3 dargestellt. Mit dem Attribut „name“ können Aktivitäten benannt werden. Die Benennung ist, bis auf wenige Ausnahmen, optional und muss nicht eindeutig sein. Da in WoG alle Nichtterminale eindeutig sein müssen, eignet sich der Name einer BPEL Aktivität nicht als eindeutiger Bezeichner des entsprechenden Nichtterminals bei WoG.

```
[01] <anyBPELActivity name="NCName"?
[02]                  suppressJoinFailure="yes|no"?>
[03]   <targets>?
[04]     <joinCondition expressionLanguage="anyURI"?>?
[05]        bool-expr
[06]     </joinCondition>
[07]     <target linkName="NCName" />+
[08]   </targets>
[09]   <sources>?
[10]     <source linkName="NCName">+
[11]       <transitionCondition expressionLanguage="anyURI"?>?
[12]         bool-expr
[13]       </transitionCondition>
[14]     </source>
[15]   </sources>
[16] </anyBPELActivity>
```

Listing 3: BPEL Standardattribute und Standardelemente, XML Struktur

Die weiteren in Listing 3 dargestellten Attribute und Elemente können nur von direkten oder indirekten Kindaktivitäten einer BPEL Flow Aktivität verwendet werden. Ihre Bedeutung und die Abbildung auf WoG werden in Kapitel 5.5.6 beschrieben. Bei der Beschreibung der XML Struktur aller anderen BPEL Aktivitäten werden diese Standardattribute- und Elemente verkürzt als „standard-attributes“ und „standard-elements“ aufgeführt.

5.2. Aufbau der Transformation

In Kapitel 3 wurden die einzelnen WoG Aktivitäten und Konstrukte vorgestellt. Ein Prozessmodell besteht im Allgemeinen aus einer Kombination von vielen Aktivitäten und unterschiedlichen Konstrukten. An Beispielen wurde gezeigt, wie man Verzweigung und Zusammenführung (Kapitel 3.2.8) oder auch Sequenz und bedingter Kontrollfluss (Kapitel 3.2.9) zusammen anwenden kann.

Einzelne BPEL Aktivitäten können in den meisten Fällen relativ einfach anhand der in Kapitel 3 beschriebenen WoG Aktivitäten und Konstrukte transformiert werden. Die Herausforderung liegt nun darin, dass ein BPEL Prozess aus einer beliebigen Kombination von Aktivitäten bestehen kann, die zusätzlich ineinander verschachtelt sind. Die Blockstruktur von BPEL ist in Abbildung 35 an einem kleinen Beispiel dargestellt. Betrachtet man die Sequenz SEQ1, dann hat sie drei Kindaktivitäten. Diese Kindaktivitäten können selbst wieder verschachtelt weitere Kindaktivitäten enthalten. Am Beispiel gilt dies für den Scope SCO2.

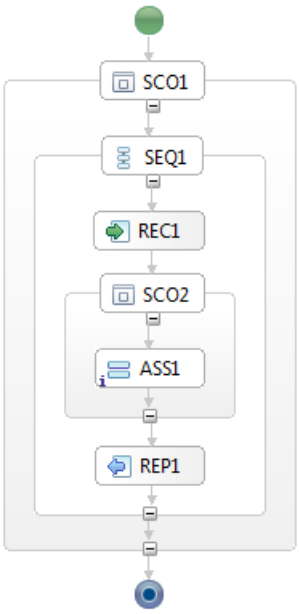


Abbildung 35: BPEL Prozess

Die Blockstruktur von BPEL ermöglicht es, die Transformation von BPEL nach WoG so zu formulieren, dass sie modular aufgebaut und algorithmisch gut zu beschreiben ist. Jede BPEL Aktivität wird auf eine Menge von WoG Aktivitäten und WoG Konstrukten abgebildet. Die entsprechende Regelmenge wird über genau einen Platzhalter aktiviert und sie erzeugt zum Schluss ebenfalls einen Platzhalter, der ihr Ende markiert. Bei der Abbildung einer BPEL Aktivität auf ihre WoG Regelmenge werden die Kinder dieser BPEL Aktivität nicht mit abgebildet. Sie werden lediglich durch die speziellen Platzhalter repräsentiert, welche die Regelmengen der Kindelemente aktivieren bzw. deren Ende markieren. Die Kindelemente selbst werden in getrennten Regelmengen modelliert. Zu jeder BPEL Aktivität gibt es genau eine Regelmenge. Die Regelmengen der einzelnen BPEL Aktivitäten werden durch ihre Beginn- und Endeplatzhalter miteinander verknüpft.

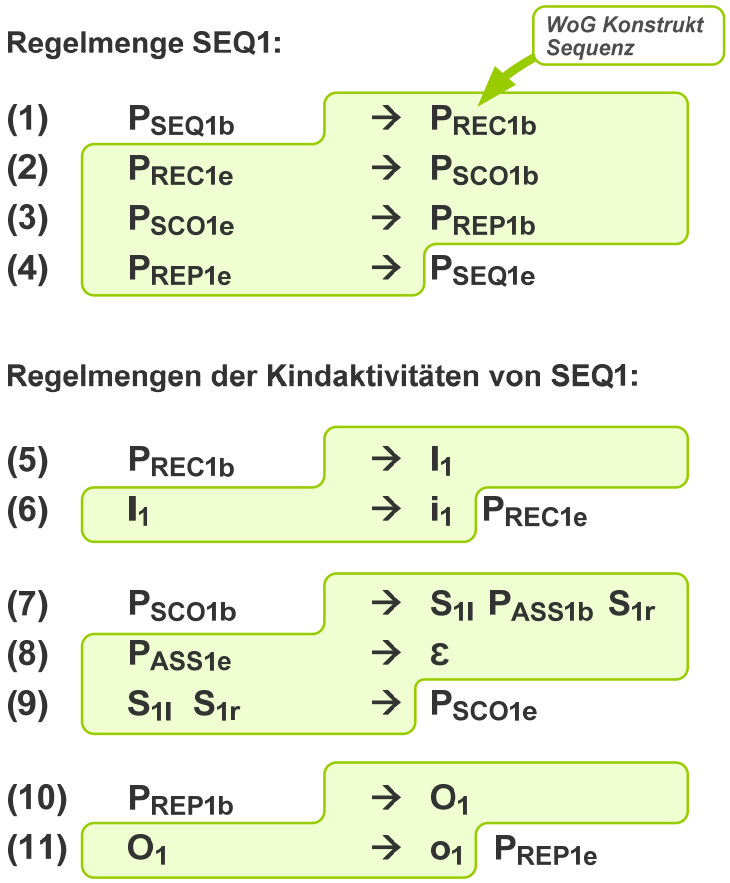


Abbildung 36: BPEL Sequence in WoG

Abbildung 36 zeigt, wie die BPEL Sequenz SEQ1 aus dem vorigen BPEL Beispielprozess auf eine WoG Regelmenge abgebildet wird. Der Platzhalter P_{SEQ1b} ist der Beginnplatzhalter dieser Regelmenge, über ihn kann sie aktiviert werden. Die Regeln dieser Regelmenge können erst dann angewendet werden, wenn dieser Platzhalter erzeugt wurde. P_{SEQ1e} ist der Endeplatzhalter dieser Regelmenge. Er wird erst dann erzeugt, wenn die Regelmenge durchlaufen wurde. Der Endeplatzhalter signalisiert nach außen, dass die entsprechende Regelmenge abgearbeitet wurde.

Die dargestellte Sequenz SEQ1 besteht aus den drei Kindaktivitäten REC1, SCO1 und REP1. Die erste Aktivität der Sequenz REC1 wird in Regel (1) aktiviert, indem durch Erzeugung des Platzhalters P_{REC1b} ihre Regelmenge aktiviert wird. Nach Beendigung der Aktivität REC1 wird die Aktivität SCO1 aktiviert. Dies wird in Regel (2) dargestellt. Der Platzhalter P_{REC1e}, der erst zum Ende der Aktivität REC1 erzeugt wird, erzeugt wiederum den Platzhalter P_{SCO1b}.

Die Beginn- und Endeplatzhalter repräsentieren somit eine Schnittstelle für die Regelmengen einzelner BPEL Aktivitäten. Die Verknüpfung der Regelmengen untereinander erfolgt ausschließlich über diese Schnittstellen. Diesen Zusammenhang stellt Abbildung 37 dar. Auf der linken Seite ist die Sequenz SEQ1 dargestellt. Sie bietet nach außen zwei Schnittstellen an, über welche sie zum einen aktiviert werden kann (P_{SEQ1b}) und zum anderen ihr Ende signalisiert (P_{SEQ1e}). Die Kindaktivitäten der Sequenz werden ebenso ausschließlich über ihre Schnittstellen angesprochen. Auf der rechten Seite ist die Regelmenge der Sequenz dargestellt. Die Regeln innerhalb des Kastens modellieren die Sequenz selbst. Die außerhalb des Kastens dargestellten Regeln werden in den Regelmengen anderer Aktivitäten verwendet, um die Sequenz zu aktivieren (… → P_{SEQ1b}) und um nach ihrer Beendigung weiteren Kontrollfluss zu modellieren (P_{SEQ1e} → ….).

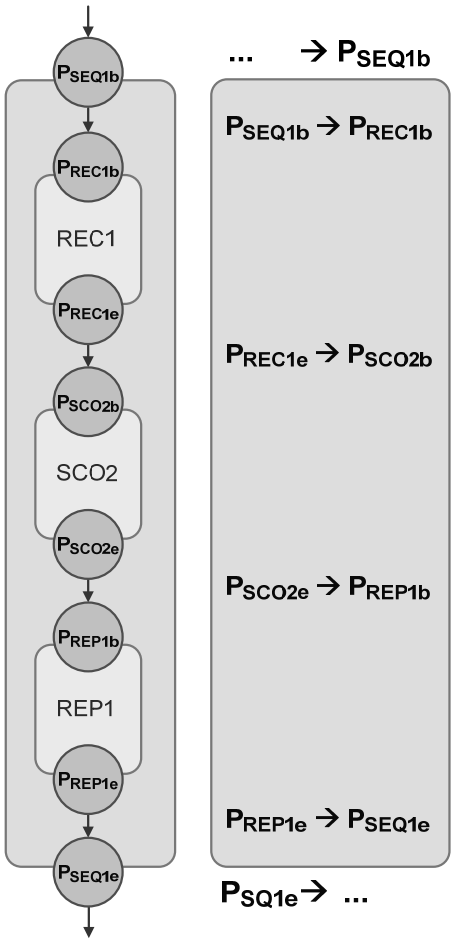


Abbildung 37: Platzhalter als Schnittstellen

5.3. BPEL Process

Ein BPEL Prozess wird durch ein „process“ Element definiert. Die Struktur eines BPEL Prozesses ist in Listing 4 dargestellt. Er repräsentiert einen Scope für alle Aktivitäten des Prozessmodells. Dieser sogenannte Prozessscope ähnelt der BPEL Aktivität Scope. Die Abbildung einer BPEL Scope Aktivität auf WoG wird in Kapitel 5.5.8 beschrieben. In diesem Kapitel werden lediglich die Attribute und Elemente eines Prozesses betrachtet, die den Prozess von einer BPEL Scope Aktivität unterscheiden.

Ein BPEL „process“ Element wird abgebildet auf einen WoG Scope, der durch das Startsymbol der entsprechenden Grammatik erzeugt wird. Im Folgenden wird für die einzelnen Attribute und Elemente eine BPEL Prozesses beschrieben, wie sie auf WoG abgebildet werden.

```
[01] <process name="NCName"
[02]          targetNamespace="anyURI"
[03]          queryLanguage="anyURI"?
[04]          expressionLanguage="anyURI"?
[05]          suppressJoinFailure="yes|no"?
[06]          exitOnStandardFault="yes|no"?
[07]          xmlns="http://docs.oasis-
[08]                open.org/wsbpel/2.0/process/executable">
[09]   <extensions>?
```

```
[10]     <extension namespace="anyURI" mustUnderstand="yes|no" />+
[11]   </extensions>
[12]   <import namespace="anyURI"?
[13]           location="anyURI"?
[14]           importType="anyURI" />*
[15]   <partnerLinks>?        …   </partnerLinks>
[16]   <messageExchanges>?    …   </messageExchanges>
[17]   <variables>?           …   </variables>
[18]   <correlationSets>?     …   </correlationSets>
[19]   <faultHandlers>?       …   </faultHandlers>
[20]   <eventHandlers>?       …   </eventHandlers>
[21]   activity
[22] </process>
```

Listing 4: BPEL Process, XML Struktur

name

Das Attribut „name“ benennt ein Prozessmodell. Dieses Attribut wird in WoG als Name der Grammatik übernommen.

targetNamespace, queryLanguage, expressionLanguage

Das Attribut „targetNamespace“ definiert den Namespace des beschriebenen Prozesses. Über die Attribute „queryLanguage“ und „expressionLanguage“ kann angegeben werden, welche Sprache im Prozess für die Formulierung von Ausdrücken und Abfragen benutzt werden. Diese Attribute werden unverändert in das WoG Startsymbol übernommen.

Extensions

BPEL ist erweiterbar. Über das Element „extensions“ werden die in einem Prozessmodell verwendeten Erweiterungen deklariert. Mit dem Attribut „mustUnderstand“ wird für jede Erweiterung angegeben, ob sie bei der Ausführung unterstützt werden muss oder ob sie ignoriert werden darf. Das Element „extensions“ wird unverändert in das WoG Startsymbol übernommen.

Das Attribut „mustUnderstand“ hat Auswirkungen auf die Übersetzung eines BPEL Prozesses auf WoG. Alle Erweiterungen, bei denen das Attribut „mustUnderstand“ den Wert „yes“ hat, müssen nach WoG übersetzt werden. Alle anderen Erweiterungen dürfen bei der Übersetzung ignoriert werden.

Import

Mit dem Element „import“ können zusätzliche Dokumente in den BPEL Prozess importiert werden. Dies können beispielsweise WSDL oder XML Schema Dokumente sein. Dieses Element wird unverändert in das WoG Startsymbol übernommen.

5.3.1. Zusammenfassung

Eine Workflow Grammatik besitzt immer genau ein Startsymbol. Jede Ableitung eines Wortes beginnt mit diesem Startsymbol. Aus diesem Grund werden alle globalen Definitionen für einen Prozess in diesem WoG Startsymbol hinterlegt. Listing 5 zeigt den Aufbau eines WoG Startsymbols.

```
[01] <nonTerminal xsi:type="tStartSymbol"
[02]              id="NCName"
[03]              name="NCName"?
[04]              targetNamespace="anyURI"
[05]              queryLanguage="anyURI"?
[06]              expressionLanguage="anyURI"?>
[07]   <extensions>?
[08]     <extension namespace="anyURI" mustUnderstand="yes|no" />+
[09]   </extensions>
[10]   <import namespace="anyURI"?
[11]           location="anyURI"?
[12]           importType="anyURI" />*
[13] </nonTerminal>
```

Listing 5: WoG Startsymbol, XML Struktur

Listing 6 zeigt ein Beispiel für einen BPEL Prozess PROC1. Dieser Prozess wird in WoG auf Regelmenge 23 abgebildet. Regel (1) beschreibt den Beginn jeder Ableitung dieser WoG Grammatik. Das Startsymbol beinhaltet alle globalen Definitionen des Prozessmodells. In Regel (3) wird der Prozessscope S_0 erzeugt. Durch das Erzeugen des Beginnplatzhalters P_{SEQ1b} der Kindaktivität SEQ1 kann diese im nächsten Schritt aktiviert werden. Regel (4) und (5) beschreiben, dass nach der Beendigung der Kindaktivität SEQ1 der Prozessscope S_0 beendet wird. Regel (2) sagt aus, dass damit auch der gesamte Prozess beendet wird.

```
[01] <process name="PROC1"
[02]          targetNamespace="www.example.com" >
[03]   <import location="proc1.wsdl"
[04]           importType="http://schemas.xmlsoap.org/wsdl/" />
[05]   <sequence name="SEQ1">       …   </sequence>
[06] </process>
```

Listing 6: BPEL Process PROC1

(1) $S \rightarrow P_{PROC1b}$

(2) $P_{PROC1e} \rightarrow \varepsilon$

(3) $P_{PROC1b} \rightarrow S_{0l}\ P_{SEQ1b}\ S_{0r}$

(4) $P_{SEQ1e} \rightarrow \varepsilon$

(5) $S_{0l}\ S_{0r} \rightarrow P_{PROC1e}$

Regelmenge 23: WoG Regelmenge für BPEL Process PROC1

5.4. BPEL Basisaktivitäten

BPEL unterscheidet zwischen Basisaktivitäten und strukturierten Aktivitäten. Im Gegensatz zu strukturierten Aktivitäten können Basisaktivitäten keine Kindaktivitäten haben. Basisaktivitäten repräsentieren Aktionen, die während der Prozessausführung durchgeführt werden, während die strukturierten Aktivitäten im wesentlichen Kontrollfluss zwischen ihren Kindaktivitäten beschreiben.

5.4.1. Invoke

Eine BPEL Invoke Aktivität beschreibt den Aufruf eines Web Services. Diese Aktivität ist immer von einem impliziten Scope umgeben. Eine BPEL Invoke Aktivität wird auf einen WoG Scope mit einer darin enthaltenen WoG Call Aktivität und gegebenenfalls noch weitere WoG Konstrukte und Aktivitäten abgebildet. Listing 7 zeigt den Aufbau einer BPEL Invoke Aktivität. Im Folgenden wird für alle hier dargestellten Elemente und Attribute beschrieben, wie sie auf WoG abgebildet werden.

```
[01] <invoke partnerLink="NCName"
[02]         portType="QName"?
[03]         operation="NCName"
[04]         inputVariable="BPELVariableName"?
[05]         outputVariable="BPELVariableName"?
[06]         standard-attributes>
[07]    standard-elements
[08]   <correlations>?
[09]     <correlation set="NCName"
[10]                  initiate="yes|join|no"?
[11]                  pattern="request|response|request-response"? />+
[12]   </correlations>
[13]   <catch faultName="QName"?
[14]          faultVariable="BPELVariableName"?
[15]          faultMessageType="QName"?
[16]          faultElement="QName"?>*
[17]     activity
[18]   </catch>
[19]   <catchAll>?
[20]     activity
[21]   </catchAll>
[22]   <compensationHandler>?
[23]     activity
[24]   </compensationHandler>
[25]   <toParts>?
[26]     <toPart part="NCName" fromVariable="BPELVariableName" />+
[27]   </toParts>
[28]   <fromParts>?
[29]     <fromPart part="NCName" toVariable="BPELVariableName" />+
[30]   </fromParts>
[31] </invoke>
```

Listing 7: BPEL Invoke, XML Struktur

partnerLink, portType, operation

Die Attribute partnerLink, portType und operation geben an, welche Operation von welchem Web Service dieses Invoke aufruft. Diese Attribute werden in die WoG Call Aktivität übernommen.

inputVariable, outputVariable

Das Attribut „inputVariable“ gibt die Variable an, in der die Inputmessage des aufzurufenden Web Services hinterlegt ist. Das Attribut „outputVariable“ gibt die Variable an, in der die Ergebnisnachricht des Web Service Aufrufs gespeichert wird. In WoG werden die entsprechenden Variablen in die Aktivierungs- und Beendigungsregel der Call Aktivität eingefügt (siehe Kapitel 3.2.4, Datenfluss).

```
[01] <invoke name="INV1"
[02]              partnerLink="INV1_PL"
[03]              portType="ns1:BPEL_Invoke_01"
[04]              operation="process"
[05]              inputVariable="input_data"
[06]              outputVariable="output_data" >
[07] </invoke>
```

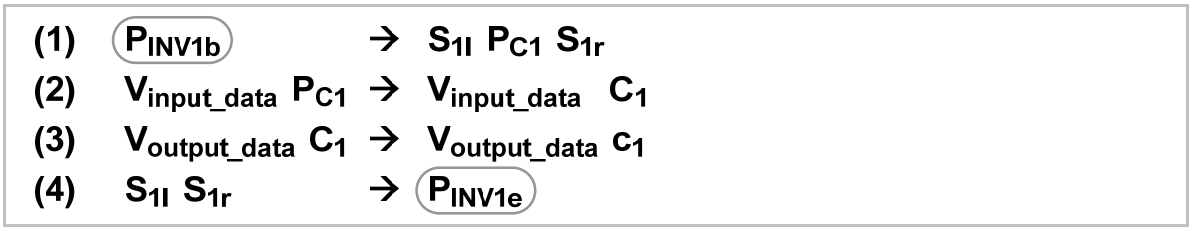
Listing 8: BPEL Invoke INV1 mit Input- und Outputvariablen

```
[01] <nonTerminal xsi:type="tCall"
[02]              id="C1"
[03]              name="INV1"
[04]              partnerLink="INV1_PL"
[05]              portType="ns1:BPEL_Invoke_01"
[06]              operation="process" >
[07] </nonTerminal>
```

Listing 9: WoG Call C_1

Listing 8 zeigt eine BPEL Invoke Aktivität INV1 mit einer Inputvariablen „input_data“ (Zeile 05) und einer Outputvariablen „output_data“ (Zeile 06). Regelmenge 24 modelliert diese BPEL Invoke Aktivität INV1 in WoG. Die in den Regeln verwendete WoG Call Aktivität C_1 ist in Listing 9 dargestellt. Wie man sieht, fehlen hier die Inputvariable „input_data“ und die Outputvariable „output_data“.

In Regel (1) der Regelmenge 24 wird aus dem Beginnplatzhalter P_{INV1b} der Scope, der in BPEL implizit vorhanden ist, explizit erzeugt. Der Scope S_1 beinhaltet einen Platzhalter P_{C1} für die Aktivität C_1. Regel (2) und (3) aktivieren und beenden die WoG Call Aktivität C_1. Die in BPEL angegebenen Input- und Outputvariablen werden hier in die Aktivierungs- bzw. Beendigungsregel integriert. Bei der Aktivierung hat die Aktivität C_1 lesenden Zugriff auf die Variable „input_data“. Bei der Beendigung hat sie schreibenden Zugriff auf die Variable „output_data“. In Regel (4) wird der Scope S_1 beendet und der Endeplatzhalter P_{INV1e} erzeugt.



Regelmenge 24: WoG Regelmenge für BPEL Invoke (a)

Die Ein- und Ausgabedaten einer BPEL Invoke Aktivität können alternativ auch über die Elemente „toParts“ und „fromParts“ angeben werden. Die Abbildung dieser Elemente auf WoG wird in dem entsprechenden Abschnitt erklärt.

Correlations

Das Element „correlations“ gibt in komlpexeren Kommunikationsszenarien an, wie Nachrichten und Prozessinstanzen eindeutig zugeordnet werden können. Dieses Element wird in die Call Aktivität übernommen.

catch, catchAll, compensationHandler

Eine BPEL Invoke Aktivität ist immer von einem impliziten BPEL Scope umgeben. Die Elemente „catch“, „catchAll“ und „compensationHandler“ gehören zu diesem impliziten Scope. Ihre Abbildung auf WoG wird im Kapitel 5.5.8 erklärt.

fromParts, toParts

Die Verwendung der oben beschriebenen Attribute „inputVariable“ und „outputVariable“ setzt voraus, dass die angegebenen Variablen den entsprechenden WSDL MessageType besitzen. Eine WSDL Message besteht immer aus einem oder mehreren „Parts“. Eine WSDL Message, die mehr als ein Part besitzt, nennt man auch Multipart Message. Jeder dieser Parts besitzt einen XML Schema Datentyp. Bei der Verwendung von WSDL Multipart Messages kann es sein, dass die Daten, die in den einzelnen Parts enthalten sein sollen, in unterschiedlichen BPEL Variablen vorliegen. Für die Verwendung einer Invoke Aktivität ist es in so einem Fall notwendig, eine zusätzliche Variable vom benötigten MessageType (Multipart) zu deklarieren und die Werte der einzelnen Variablen mit Hilfe einer BPEL Assign Aktivität in die entsprechenden Parts dieser neuen Variablen zu kopieren.

Die Elemente „toParts“ und „fromParts“ sind Komfortfunktionen, die es dem Modellierer vereinfachen, mit WSDL Multipart Messages umzugehen. Die Konstruktion der Inputnachricht bzw. die Dekonstruktion der Outputnachricht muss nicht wie oben beschrieben explizit modelliert werden. Das Element „toParts“ wird verwendet, um eine temporäre und transparente Variable vom Typ der entsprechenden Nachricht zu erstellen und die einzelnen Parts dieser neuen Variablen mit den Werten der angegebenen Variablen zu befüllen. Das Element „fromParts“ wird verwendet, um die Ergebnisnachricht in einer temporären und transparenten MessageType Variablen zwischenzuspeichern und die Werte der einzelnen Parts in die angegebenen Variablen zu kopieren.

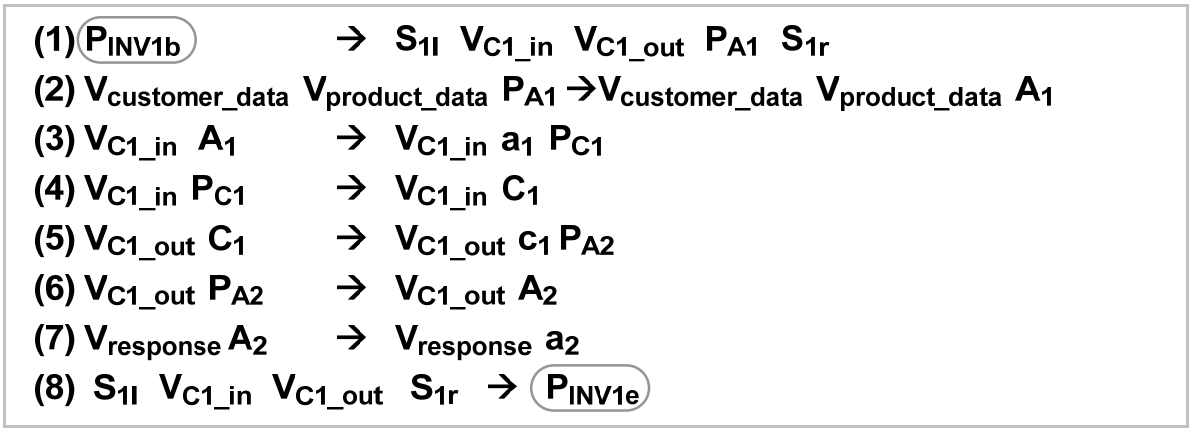
In WoG wird die Konstruktion der Eingabenachricht bzw. die Dekonstruktion der Ausgabenachricht immer explizit modelliert. Wird in einer BPEL Invoke Aktivität das Element „toParts“ benutzt, so wird in WoG eine zusätzliche Variable vom Typ der Eingabenachricht deklariert. Direkt vor der WoG Call Aktivität wird eine zusätzliche WoG Assign Aktivität eingefügt, welche die Kopiervorgänge ausführt, die durch das „toParts“ beschrieben werden. Wird in einer BPEL Invoke Aktivität das Element „fromParts“ benutzt, so wird in WoG eine zusätzliche Variable vom Typ der Ausgabenachricht deklariert. Direkt nach der WoG Call Aktivität wird eine zusätzliche WoG Assign Aktivität eingefügt, welche die durch das „fromParts“ beschriebenen Kopiervorgänge ausführt.

```
[01] <invoke name="INV1"
[02]             partnerLink="INV1_PL"
[03]             operation="process"
[04]             portType="tns:BPEL_Invoke_01">
[05]   <toParts>
[06]     <toPart part="customer"
[07]                  fromVariable="customer_data" />
[08]     <toPart part="product"
[09]                  fromVariable="product_data" />
[10]   </toParts>
[11]   <fromParts>
[12]     <fromPart part="payload"
[13]                   toVariable="response" />
[14]   </fromParts>
[15] </invoke>
```

Listing 10: BPEL Invoke mit toParts und fromParts

In Listing 10 ist ein BPEL Invoke mit den Elementen „toParts“ und „fromParts“ dargestellt. Diese BPEL Invoke Aktivität wird in WoG auf die Regelmenge 25 abgebildet. Die in dieser Regelmenge enthaltene WoG Call Aktivität C_1 wurde bereits in Listing 9 beschrieben. Bei der Abbildung einer BPEL Invoke Aktivität auf WoG entsteht immer genau eine Call Aktivität. Diese Aktivität ist unabhängig davon, ob das BPEL Invoke den Datenzugriff über die Attribute „inputVariable“ und „outputVariable“ oder über die Elemente „toParts“ und „fromParts“ spezifiziert.

In Regel (1) der Regelmenge 25 wird aus dem Beginnplatzhalter P_{INV1b} der Scope S_1, der in BPEL implizit vorhanden ist, explizit erzeugt. Der Scope S_1 beinhaltet zwei Variablen, die in BPEL nicht vorhanden sind. In V_{C1_in} werden die Eingabedaten für das Call C_1 abgelegt. In V_{C1_out} werden die Ausgabedaten der Call Aktivität C_1 abgelegt. Zusätzlich zu den Variablen enthält der Scope einen Platzhalter P_{A1} für die Assign Aktivität A_1. Diese Aktivität führt die in „toParts“ beschriebenen Kopiervorgänge aus. Im Beispiel werden die Werte der Variablen $V_{customer\_data}$ und $V_{product\_data}$ in die entsprechenden Parts der Variablen V_{C1_in} kopiert (Regeln (2) und (3)). Nach Beendigung der Assign Aktivität A_1 wird die Call Aktivität C_1 aktiviert, sie liest die Eingabedaten aus der Variablen V_{C1_in}. In Regel (5) wird C_1 beendet, die Ausgabedaten werden in die Variable V_{C1_out} geschrieben und es wird der Platzhalter P_{A2} für das nachfolgende Assign A_2 erzeugt. In Regel (6) und (7) wird beschrieben, dass das Assign A_2 die Daten aus dem Part der Variable V_{C1_out} in die Variable $V_{response}$ kopiert. Regel (8) löscht den Scope S_1 und die darin lebenden Variablen und erzeugt den Endeplatzhalter P_{INV1e}.



Regelmenge 25: WoG Regelmenge für BPEL Invoke (b)

5.4.1.1. Zusammenfassung

Eine BPEL Invoke Aktivität wird auf eine WoG Regelmenge abgebildet, welche einen Scope, die Aktivierung und die Beendigung einer Call Aktivität sowie gegebenenfalls weitere Assign Aktivitäten modelliert. Der Aufbau der WoG Regelmenge wurde bereits in Regelmenge 24 und Regelmenge 25 gezeigt. Der allgemeine Aufbau eines Nichtterminals vom Aktivitätstyp Call wird in Listing 11 dargestellt. Durch das Attribut „type“ in Zeile 1 wird der Typ des Nichtterminals festgelegt, in diesem Fall ist es ein Call. Jedes Nichtterminal wird durch das Attribut „id“ (Zeile 2) eindeutig identifiziert. Zusätzlich kann bei jedem Nichtterminal das Attribut „Name“ gesetzt werden (Zeile 3). Der Name einer BPEL Aktivität wird auf dieses Attribut abgebildet. Alle weiteren Attribute und Kindelemente beschreiben die Web Service Kommunikation und werden unverändert von BPEL übernommen.

```
[01] <nonTerminal xsi:type="tCall"
[02]              id="NCName"
[03]              name="NCName"?
[04]              partnerLink="NCName"
[05]              portType="QName"?
[06]              operation="NCName" >
[07]   <correlations>?
[08]     <correlation set="NCName"
[09]                  initiate="yes|join|no"?
[10]                  pattern="request|response|request-response"? />+
[11]   </correlations>
[12] </nonTerminal>
```

Listing 11: WoG Call, XML Struktur

5.4.2. Receive

Ein BPEL Receive Aktivität beschreibt den Empfang einer Nachricht über eine Web Service Schnittstelle, die der Prozess nach außen hin anbietet. Eine BPEL Receive Aktivität wird auf eine WoG In Aktivität mit genau einem Interface und gegebenenfalls noch weitere WoG Konstrukte und Aktivitäten abgebildet. Listing 12 zeigt den Aufbau einer BPEL Receive Aktivität. Im Folgenden wird für alle hier dargestellten Elemente und Attribute beschrieben, wie sie auf WoG abgebildet werden.

```
[01] <receive partnerLink="NCName"
[02]          portType="QName"?
[03]          operation="NCName"
[04]          variable="BPELVariableName"?
[05]          createInstance="yes|no"?
[06]          messageExchange="NCName"?
[07]          standard-attributes >
[08]   standard-elements
[09]   <correlations>?
[10]     <correlation set="NCName"
[11]                  initiate="yes|join|no"? />+
[12]   </correlations>
[13]   <fromParts>?
[14]     <fromPart part="NCName"
[15]               toVariable="BPELVariableName" />+
[16]   </fromParts>
[17] </receive>
```

Listing 12: BPEL Receive, XML Struktur

partnerLink, portType, operation

Die Attribute partnerLink, portType und operation geben an, über welche Operation von welchem Web Service dieses Receive Nachrichten empfangen kann. Diese Attribute werden in die In Aktivität übernommen.

Variable

Das Attribut „variable“ gibt die Variable an, in der die von der Receive Aktivität empfangene Nachricht abgespeichert wird. In WoG wird diese Variable in die Beendigungsregel der In Aktivität eingefügt.

```
[01] <receive name="REC1"
[02]              partnerLink="REC1_PL"
[03]              portType="ns1:BPEL_Receive_01"
[04]              operation="process"
[05]              variable="output_data">
[06] </receive>
```

Listing 13: BPEL Receive Aktivität REC1 mit Variable

```
[01] <nonTerminal xsi:type="tIn"
[02]             id="I1"
[03]             name="REC1"
[04]             partnerLink="REC1_PL"
[05]             portType="ns1:BPEL_Receive_01"
[06]             operation="process" >
[07] </nonTerminal>
```

Listing 14: WoG In Aktivität I_1

Listing 13 zeigt eine BPEL Receive Aktivität REC1 mit einer Variablen „output_data“ (Zeile 05). Regelmenge 26 modelliert diese BPEL Receive Aktivität REC1 in WoG. Die in den Regeln verwendete WoG In Aktivität I_1 ist in Listing 14 dargestellt, die Variable „output_data“ ist dort nicht aufgeführt. Der Zugriff auf die Variable „output_data“ wird in WoG explizit in den Regeln modelliert.

In Regel (1) der Regelmenge 26 aktiviert der Beginnplatzhalter P_{REC1b} die WoG In Aktivität I_1. In Regel (2) wird die Aktivität I_1 beendet und der Endeplatzhalter P_{REC1e} erzeugt. Die in BPEL angegebene Variable „output_data“ (Listing 13, Zeile 5) wird in diese Beendigungsregel integriert, I_1 hat schreibenden Zugriff auf die Variable „output_data“.

Regelmenge 26: WoG Regelmenge für BPEL Receive (a)

Die Ausgabedaten einer BPEL Receive Aktivität können alternativ auch über das Element „fromParts“ angeben werden. Die Abbildung dieses Elementes wird in dem entsprechenden Abschnitt erklärt.

createInstance

Prozessinstanzen werden in BPEL immer durch den Empfang von Nachrichten erzeugt. Mit dem Attribut „createInstance“ kann bei entsprechenden Aktivitäten (Receive und Pick) angegeben werden, ob der Empfang einer Nachricht eine neue Prozessinstanz erzeugt. Dieses Attribut wird in WoG übernommen.

messageExchange

In BPEL kann eine Receive Aktivität mit einer Reply Aktivität kombiniert werden, um das Empfangen einer Nachricht und das spätere Senden einer Antwort dieser Nachricht zu modellieren. Von außen entspricht dies einem synchronen Aufruf einer vom Prozess angebotenen Operation. Das Attribut „messageExchange“ wird benutzt, um in einem solchen Fall eine Receive Aktivität mit einer Reply Aktivität zu verknüpfen. Dieses Attribut wird in WoG übernommen, um eine In Aktivität mit einer dazugehörenden Out Aktivität zu korrelieren.

Correlations

Das Element „correlations“ gibt in komplexeren Kommunikationsszenarien an, wie Nachrichten und Prozessinstanzen eindeutig zugeordnet werden können. Dieses Element wird in die WoG In Aktivität übernommen.

fromParts

Mit Hilfe des Elementes „fromParts“ können die Parts der empfangenen Nachricht direkt auf unterschiedliche Variablen abgebildet werden. In Kapitel 5.4.1 wurde dieses Element bereits detailliert beschreiben.

Wird in einer BPEL Receive Aktivität das Element „fromParts“ benutzt, so wird in WoG eine zusätzliche Variable vom entsprechenden Nachrichtentyp deklariert. Direkt nach der WoG In Aktivität wird eine WoG Assign Aktivität eingefügt, welche die durch das „fromParts“ beschriebenen Kopiervorgänge ausführt.

Listing 15 zeigt eine BPEL Receive Aktivität REC1 mit dem Element „fromParts“. In WoG wird diese Receive Aktivität auf die Regelmenge 27 abgebildet. Die in dieser Regelmenge enthaltene WoG In Aktivität I_1 wurde bereits in Listing 14 beschrieben. Bei der Abbildung einer BPEL Receive Aktivität auf WoG entsteht immer genau eine In Aktivität, unabhängig davon, ob das BPEL Receive den Datenzugriff über das Attribut „variable“ oder über das Element „fromParts“ spezifiziert.

```
[01] <receive name="REC1"
[02]                partnerLink="REC1_PL"
[03]                portType="ns1:BPEL_Receive_01"
[04]                operation="process" >
[05]   <fromParts>
[06]     <fromPart part="payload"
[07]                    toVariable="customer_data" />
[08]   </fromParts>
[09] </receive>
```

Listing 15: BPEL Receive Aktivität REC1 mit fromParts

In Regel (1) der Regelmenge 27 wird mit Hilfe des Beginnplatzhalters P_{REC1b} ein Scope S_1, die Variable V_{I1_out} sowie ein Platzhalter P_{I1} für die WoG In Aktivität erzeugt. Die Variable V_{I1_out} gibt es in der BPEL Aktivität nicht. Sie wird in WoG benötigt, um die Ausgabedaten der In Aktivität I_1 abzulegen. Den Scope S_1 gibt es in BPEL ebenfalls nicht, er dient in WoG lediglich als Lebensraum für die neu erzeugte Variable V_{I1_out}. In Regel (2) wird die In Aktivität I_1 aktiviert. In Regel (3) wird I_1 beendet, die Ausgabedaten werden in die Variable V_{I1_out} geschrieben und es wird der Platzhalter P_{A1} für das nachfolgende Assign A_1 erzeugt. Regel (4) und (5) beschreiben, dass das Assign A_1 die Daten aus dem Part der Variable V_{I1_out} in die Variable $V_{customer\_data}$ kopiert. In Regel (6) wird der Scope S_1 und die darin lebende Variable V_{I1_out} gelöscht sowie der Endeplatzhalter P_{REC1e} erzeugt.

(1)	P_{REC1b}	→	$S_{1l}\ V_{I1\_out}\ P_{I1}\ S_{1r}$
(2)	P_{I1}	→	I_1
(3)	$V_{I1\_out}\ I_1$	→	$V_{I1\_out}\ i_1\ P_{A1}$
(4)	$V_{I1\_out}\ P_{A1}$	→	$V_{I1\_out}\ A_1$
(5)	$V_{customer\_data}\ A_1$	→	$V_{customer\_data}\ a_1$
(6)	$S_{1l}\ V_{I1\_out}\ S_{1r}$	→	P_{REC1e}

Regelmenge 27: WoG Regelmenge für BPEL Receive (b)

5.4.2.1. Zusammenfassung

Eine BPEL Receive Aktivität wird auf eine WoG Regelmenge abgebildet, welche die Aktivierung und die Beendigung einer In Aktivität sowie gegebenenfalls einen Scope und eine Assign Aktivität modelliert. Der Aufbau der Wog Regelmenge wurde bereits in Regelmenge 26 und Regelmenge 27 gezeigt. Der allgemeine Aufbau eines Nichtterminals vom Aktivitätstyp In wird in Listing 16 dargestellt. Durch das Attribut „type“ in Zeile 1 wird der Typ des Nichtterminals festgelegt, in diesem Fall ist es ein In. Die Attribute „id“ (Zeile 2) und „name“ (Zeile (3) kommen bei allen Nichtterminalen vor und wurden bereits in Kapitel 5.4.1 beschrieben. Eine In Aktivität enthält mindestens ein Element „interface“ (Zeile 4). Jedes „interface“ Element beschreibt eine Schnittstelle, über welche die In Aktivität Nachrichten empfangen kann. Eine In Aktivität kann mehrere Schnittstellen anbieten, sie empfängt jedoch immer genau eine Nachricht. Die Verwendung einer In Aktivität mit mehreren „interface“ Elementen wird in Kapitel 5.5.5 gezeigt. Die Attribute und Kindelemente eines „interface“ Elementes beschreiben die Web Service Kommunikation und werden unverändert von BPEL übernommen.

```
[01] <nonTerminal xsi:type="tIn"
[02]              id="NCName"
[03]              name="NCName"? >
[04]   <interface partnerLink="NCName"
[05]          portType="QName"?
[06]          operation="NCName"
[07]          createInstance="yes|no"?
[08]          messageExchange="NCName"? >+
[09]     <correlations>?
[10]       <correlation set="NCName"
[11]                    initiate="yes|join|no"? />+
[12]     </correlations>
[13]   </interface>
[14] </nonTerminal>
```

Listing 16: WoG In, XML Struktur

5.4.3. Reply

```
[01] <reply partnerLink="NCName"
[02]        portType="QName"?
[03]        operation="NCName"
[04]        variable="BPELVariableName"?
[05]        faultName="QName"?
[06]        messageExchange="NCName"?
[07]        standard-attributes>
[08]    standard-elements
[09]    <correlations>?
[10]       <correlation set="NCName"
[11]                    initiate="yes|join|no"? />+
[12]    </correlations>
[13]    <toParts>?
[14]       <toPart part="NCName"
[15]                fromVariable="BPELVariableName" />+
[16]    </toParts>
[17] </reply>
```

Listing 17: BPEL Reply, XML Struktur

Eine BPEL Reply Aktivität beschreibt das Senden einer Antwortnachricht. Diese bezieht sich immer auf ein vorangegangenes Receive oder Pick. Eine BPEL Reply Aktivität wird auf eine WoG Out

Aktivität und gegebenenfalls noch weitere WoG Konstrukte und Aktivitäten abgebildet. Listing 17 zeigt den Aufbau einer BPEL Reply Aktivität. Im Folgenden wird für alle hier dargestellten Elemente und Attribute beschrieben, wie sie auf WoG abgebildet werden.

partnerLink, portType, operation

Die Attribute partnerLink, portType und operation geben an, zu welcher Operation von welchem Web Service dieses Reply eine Antwortnachricht sendet. Diese Attribute werden in die WoG Out Aktivität übernommen.

Variable

Das Attribut „variable“ gibt die Variable an, in der die von der Reply Aktivität zu sendende Nachricht abgespeichert ist. In WoG wird diese Variable in die Aktivierungsregel der Out Aktivität eingefügt.

```
[01] <reply name="REP1"
[02]             partnerLink="REP1_PL"
[03]             portType="ns1:BPEL_Reply_01"
[04]             operation="process"
[05]             variable="input_data" >
[06] </reply>
```

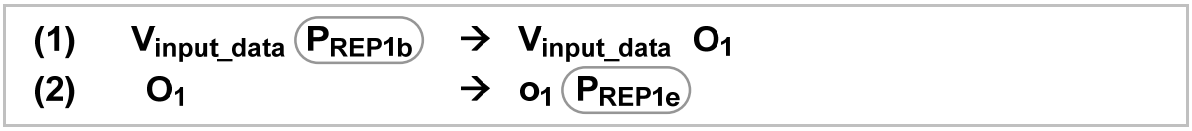
Listing 18: BPEL Reply mit Variable

```
[01] <nonTerminal xsi:type="tOut"
[02]              id="O1"
[03]              name="REP1"
[04]              partnerLink="REP1_PL"
[05]              portType="ns1:BPEL_Reply_01"
[06]              operation="process" >
[07] </nonTerminal>
```

Listing 19: WoG Out

Listing 18 zeigt eine BPEL Reply Aktivität REP1 mit einer Variablen „input_data“ (Zeile 05). Regelmenge 28 modelliert diese BPEL Reply Aktivität REP1 in WoG. Die in den Regeln verwendete WoG Out Aktivität O_1 ist in Listing 19 dargestellt, die Variable „input_data“ ist dort nicht aufgeführt. Der Zugriff auf die Variable „input_data“ wird in WoG explizit in den Regeln modelliert.

In Regel (1) der Regelmenge 28 aktiviert der Beginnplatzhalter P_{REP1b} die WoG Out Aktivität O_1. Bei dieser Aktivierung hat die Aktivität O_1 lesenden Zugriff auf die Variable „input_data“. In Regel (2) wird die Aktivität O_1 beendet und der Endeplatzhalter P_{REP1e} erzeugt.



Regelmenge 28: WoG Regelmenge für BPEL Reply (a)

Die Eingabedaten einer BPEL Reply Aktivität können alternativ auch über das Element „toParts“ angeben werden. Die Abbildung dieses Elementes wird in dem entsprechenden Abschnitt erklärt.

faultName

Ist das Attribut „faultName“ nicht angegeben, dann beschreibt die Reply Aktivität das Senden einer regulären Antwortnachricht. Ist das Attribut „faultName“ dagegen angegeben, dann sendet dieses Reply eine Fehlernachricht. Dieses Attribut wird in die WoG Out Aktivität übernommen.

messageExchange

Das Attribut „messageExchange“ wurde bereits in Kapitel 5.4.3 beschrieben. Dieses Attribut wird in die WoG Out Aktivität übernommen.

Correlations

Das Element „correlations“ gibt in komlpexeren Kommunikationsszenarien an, wie Nachrichten und Prozessinstanzen eindeutig zugeordnet werden können. Dieses Element wird in die WoG In Aktivität übernommen.

toParts

Mit Hilfe des Elementes „toParts“ können die unterschiedlichen Variablen direkt auf die Parts der zu sendenden Nachricht abgebildet werden. In Kapitel 5.4.1 wurde dieses Element bereits detailliert beschreiben.

Wird in einer BPEL Reply Aktivität das Element „toParts“ benutzt, so wird in WoG eine zusätzliche Variable vom entsprechenden Nachrichtentyp deklariert. Direkt vor der WoG Out Aktivität wird eine zusätzliche WoG Assign Aktivität eingefügt, welche die durch das „toParts“ beschriebenen Kopiervorgänge ausführt.

Listing 20 zeigt eine BPEL Reply Aktivität REP1 mit dem Element „toParts“. In WoG wird diese Reply Aktivität auf die Regelmenge 29 abgebildet. Die in dieser Regelmenge enthaltene WoG Out Aktivität O_1 wurde bereits in Listing 19 beschrieben. Bei der Abbildung einer BPEL Reply Aktivität auf WoG entsteht immer genau eine Out Aktivität, unabhängig davon, ob das BPEL Reply den Datenzugriff über das Attribut „variable“ oder über das Element „toParts“ spezifiziert..

```
[01] <reply name="REP1"
[02]             partnerLink="REP1_PL"
[03]             portType="ns1:BPEL_Reply_01"
[04]             operation="process" >
[05]   <toParts>
[06]     <toPart part="customer"
[07]                  fromVariable="customer_data" />
[08]     <toPart part="product"
[09]                  fromVariable="product_data" />
[10]   </toParts>
[11] </reply>
```

Listing 20: BPEL Reply mit toParts

In Regel (1) der Regelmenge 29 wird mit Hilfe des Beginnplatzhalters P_{REP1b} ein Scope S_1, die Variable V_{O1_in} sowie ein Platzhalter P_{A1} für die Assign Aktivität A_1 erzeugt. Die Variable V_{O1_in} gibt es in der BPEL Aktivität nicht. Sie wird in WoG benötigt, um die Eingabedaten der Out Aktivität O_1 zwischenzuspeichern. Den Scope S_1 gibt es in BPEL ebenfalls nicht, er dient in WoG lediglich als Lebensraum für die neu erzeugte Variable V_{O1_in}. Regel (2) und (3) beschreiben, dass das Assign A_1 die Daten aus den Variablen $V_{customer\_data}$ und $V_{product\_data}$ in die Parts der Variablen V_{O1_in} kopiert. In Regel (3) wird zusätzlich der Platzhalter P_{O1} für die Out Aktivität O_1 erzeugt. In Regel (4) wird die Aktivität O_1 aktiviert, sie liest die Eingabedaten aus der Variablen V_{O1_in}. Regel (5) beschreibt die

Beendigung der Aktivität O_1. In Regel (6) wird der Scope S_1 und die darin lebende Variable V_{O1_in} gelöscht sowie der Endeplatzhalter P_{REP1e} erzeugt.

(1)	P_{REP1b}	→ S_{1l} V_{O1_in} P_{A1} S_{1r}
(2)	$V_{customer\_data}$ $V_{product\_data}$ P_{A1}	→ $V_{customer\_data}$ $V_{product\_data}$ A_1
(3)	V_{O1_in} A_1	→ V_{O1_in} a_1 P_{O1}
(4)	V_{O1_in} P_{O1}	→ V_{O1_in} O_1
(5)	O_1	→ o_1
(6)	S_{1l} V_{O1_in} S_{1r}	→ P_{REP1e}

Regelmenge 29: WoG Regelmenge für BPEL Reply (b)

5.4.3.1. Zusammenfassung

Eine BPEL Reply Aktivität wird auf eine WoG Regelmenge abgebildet, welche die Aktivierung und die Beendigung einer Out Aktivität sowie gegebenenfalls einen Scope und eine Assign Aktivität modelliert. Der Aufbau der Wog Regelmenge wurde bereits in Regelmenge 28 und Regelmenge 29 gezeigt. Der allgemeine Aufbau eines Nichtterminals vom Aktivitätstyp Out wird in Listing 21 dargestellt. Durch das Attribut „type“ in Zeile 1 wird der Typ des Nichtterminals festgelegt, in diesem Fall ist es ein Out. Die Attribute „id“ (Zeile 2) und „name“ (Zeile 3) kommen bei allen Nichtterminalen vor und wurden bereits in Kapitel 5.4.1 beschrieben. Das Attribut „faultName“ gibt an, ob eine reguläre Ergebnisnachricht oder eine Fehlernachricht gesendet wird. Alle weiteren Attribute und Kindelemente beschreiben die Web Service Kommunikation und werden unverändert von BPEL übernommen.

```
[01] <nonTerminal xsi:type="tOut"
[02]              id="NCName"
[03]              name="NCName"?
[04]              partnerLink="NCName"
[05]              portType="QName"?
[06]              operation="NCName"
[07]              faultName="QName"?
[08]              messageExchange="NCName"? >
[09]   <correlations>?
[10]     <correlation set="NCName"
[11]                  initiate="yes|join|no"? />+
[12]   </correlations>
[13] </nonTerminal>
```

Listing 21: WoG Out, XML Struktur

5.4.4. Assign

Eine BPEL Assign Aktivität beschreibt Datenmanipulation auf Variablen oder PartnerLinks. Eine BPEL Assign Aktivität wird auf genau eine WoG Assign Aktivität abgebildet. Listing 22 zeigt den Aufbau einer BPEL Assign Aktivität. Im Folgenden wird für alle hier dargestellten Elemente und Attribute beschrieben, wie sie auf WoG abgebildet werden.

```
[01] <assign validate="yes|no"?
[02]         standard-attributes>
[03]    standard-elements
[04]    (
[05]    <copy keepSrcElementName="yes|no"?
[06]          ignoreMissingFromData="yes|no"?>
[07]       from-spec to-spec
[08]    </copy>
[09]    |
[10]    <extensionAssignOperation>
[11]       assign-element-of-other-namespace
[12]    </extensionAssignOperation>  )+
[13] </assign>
```

Listing 22: BPEL Assign, XML Struktur

validate

Mit dem „validate“ Attribut kann angeben werden, ob die Variablen vor der Durchführung des Kopiervorgangs gegen ihr Schema validiert werden sollen. Dieses Attribut wird in die WoG Assign Aktivität übernommen.

extensionAssignOperation

BPEL ist generell erweiterbar. Das Element „extensionAssignOperation“ beschreibt, dass auch die in der Assign Aktivität enthaltene Kopiervorgänge durch weitere nicht durch BPEL spezifizierte Mechanismen definiert werden können. Die Übersetzung dieses Elementes von BPEL auf WoG ist abhängig von den konkret verwendeten Erweiterungen und kann deswegen hier nicht näher beschrieben werden.

Copy

Eine Assign Aktivität kann mehrere „copy“ Elemente enthalten, jedes dieser Elemente spezifiziert genau eine Datenmanipulation. Diese werden zur Laufzeit nacheinander abgearbeitet. Der BPEL Standard fordert, dass die Abarbeitung der „copy“ Operationen einer Assign Aktivität Eigenschaften einer Transaktion hat. Dies bedeutet im Detail, dass alle von den Datenmanipulationen betroffenen Variablen und PartnerLinks für die Laufzeit des Assigns für den restlichen Prozess nur lesend zur Verfügung stehen. Zudem wird gefordert, dass im Falle eines Fehlers während der Abarbeitung einer Assign Aktivität die Variablen und PartnerLinks auf ihren ursprünglichen Wert, den sie vor der Aktivierung der Assign Aktivität hatten, gesetzt werden.

Eine BPEL Assign Aktivität wird auf genau eine WoG Assign Aktivität abgebildet, welche ebenfalls mehrere „copy“ Elemente enthalten kann. Um den transaktionalen Charakter der Assign Aktivität sicherzustellen, wird in ihrer Aktivierungsregel der Lebensmarker des Prozesses gelöscht. In der Beendigungsregel der Assign Aktivität wird er wieder erzeugt. Dies bedeutet, dass während der Ausführung einer Assign Aktivität keine anderen Regeln angewendet werden können. Wenn die Assign Aktivität erfolgreich durchgelaufen ist, dann wird die „normale“ Beendigungsregel angewendet und die entsprechenden Variablen erhalten ihren neuen Wert. Tritt dagegen während der Ausführung der Assign Aktivität ein Fehler auf, dann wird die Assign Aktivität mit der entsprechenden Fehlerregel beendet (siehe Kapitel 3.2.9.3). Die Variablen werden hierbei nicht beschrieben.

Das Attribut „keepSrcElementName“ gibt an, ob bei einem Kopiervorgang der Elementname des Zielelementes überschrieben wird. Das Attribut „ignoreMissingFromData“ gibt an, ob BPEL Fehler vom Typ „bpel:selectionFailure“ ignoriert werden können. Beide Attribute werden unverändert in WoG übernommen.

Es gibt in BPEL mehrere Varianten, wie die einzelnen Zuweisungen beschrieben werden können. Es ist unter anderem möglich, Ausdrücke zu verwenden, einzelne Teile von Variablen zu manipulieren oder auf PartnerLinks zuzugreifen. Jede Zuweisung besteht aus einem „from“ Element und einem „to“ Element. Vergleicht man dies mit Zuweisungen in Programmiersprachen, so entspricht das „to“ Element der linken Seite einer Zuweisung und das „from“ Element der rechten Seite. Im Folgenden wird beschrieben, welche Ausprägungen der „from“ und „to“ Elemente in BPEL definiert werden und wie diese auf WoG abgebildet werden.

5.4.4.1. from

Das „from“ Element gibt an, welche Variablen gelesen werden. Daraus folgt für WoG, dass die hier verwendeten Variablen in die Aktivierungsregel der entsprechenden WoG Assign Aktivität eingefügt werden. Listing 23 zeigt den Aufbau der einzelnen Varianten eines „from“ Elementes. Im Folgenden wird für alle hier dargestellten Alternativen beschrieben, wie sie auf WoG abgebildet werden.

```
[01] <from variable="BPELVariableName"
[02]       part="NCName"?>
[03]    <query queryLanguage="anyURI"?>?
[04]       queryContent
[05]    </query>
[06] </from>
[07] |
[08] <from partnerLink="NCName"
[09]       endpointReference="myRole|partnerRole" />
[10] |
[11] <from variable="BPELVariableName"
[12]       property="QName" />
[13] |
[14] <from expressionLanguage="anyURI"?>expression</from>
[15] |
[16] <from><literal>literal value</literal></from>
[17] |
[18] <from/>
```

Listing 23: BPEL Assign, from- Varianten, XML Struktur

„from“ – Variable

Das in den Zeilen 1 bis 6 dargestellte „from“ beschreibt den lesenden Zugriff auf eine Variable. Die zu lesende Variable wird über das Attribut „variable“ angegeben. Diese Variable wird in die Aktivierungsregel der entsprechenden WoG Assign Aktivität eingefügt.

Über das Attribut „part“ und das Element „query“ kann näher spezifiziert werden, auf welchen Teil der Variablen zugegriffen wird. Das „from“ Element wird unverändert in die entsprechende WoG Assign Aktivität übernommen.

„from“ – PartnerLink

Das in den Zeilen 8 bis 9 dargestellte „from“ beschreibt den lesenden Zugriff auf die in einem PartnerLink hinterlegten Endpunktreferenzen. Über das Attribut „endpointReference“ wird

angegeben, auf welche der im PartnerLink hinterlegten Endpunktreferenzen zugegriffen wird. Das „from“ Element wird unverändert in die entsprechende WoG Assign Aktivität übernommen. Die Workflow Engine ist zur Laufzeit dafür verantwortlich, den Wert der entsprechenden Endpunktreferenz zur Verfügung zu stellen.

„from“ – Property

Das in den Zeilen 11 bis 12 dargestellte „from“ beschreibt den lesenden Zugriff auf einen Teil einer Variablen, der als „property“ definiert ist. Die zu lesende Variable wird über das Attribut „variable“ angegeben. Diese Variable wird in die Aktivierungsregel der entsprechenden WoG Assign Aktivität eingefügt.

Das Attribut „property“ gibt an, auf welchen Teil der Variablen zugegriffen wird. Das „from“ Element wird unverändert in die entsprechende WoG Assign Aktivität übernommen.

„from“ – Ausdruck

Das in Zeile 14 dargestellte „from“ beschreibt einen Ausdruck. In BPEL wird standardmäßig XPATH als Ausdruckssprache verwendet. Über das Attribut „expressionLanguage“ können auch andere Ausdruckssprachen verwendet werden. Das „from“ Element wird unverändert in die entsprechende WoG Assign Aktivität übernommen. Der Ausdruck selbst wird bei der Übersetzung in WoG analysiert, um festzustellen, auf welche Variablen der Ausdruck zugreift. Diese Variablen werden in die Aktivierungsregel der entsprechenden WoG Assign Aktivität eingefügt.

„from“ – Literal

Das in Zeile 16 dargestellte „from“ beschreibt ein Literal, also einen Wert. Dementsprechend wird auf keine Variable zugegriffen. Das „from“ Element wird unverändert in die entsprechende WoG Assign Aktivität übernommen.

„from“ – leer

Das in Zeile 18 dargestellte „from“ Element beschreibt keinen bzw. einen „null“ Wert. Es wird unverändert in die entsprechende WoG Assign Aktivität übernommen.

5.4.4.2. to

Das „to“ Element gibt an, welche Variablen beschrieben werden. Daraus folgt für WoG, dass die hier verwendeten Variablen in die Beendigungsregel der entsprechenden WoG Assign Aktivität eingefügt werden. Abgesehen davon erfolgt die Übersetzung in WoG wie im vorherigen Kapitel für das „from“ beschrieben. Listing 24 zeigt den Aufbau der einzelnen Varianten eines „to“ Elementes. Im Gegensatz zum „from“ Element wird bei dem in Zeile 8 dargestellten Zugriff auf die in einem PartnerLink hinterlegte Endpunktreferenz keine Rolle angegeben. Schreibender Zugriff ist nur auf die Endpunktreferenz der Partnerrolle möglich.

```
[01] <to variable="BPELVariableName"
[02]     part="NCName"?>
[03]    <query queryLanguage="anyURI"?>?
[04]       queryContent
[05]    </query>
[06] </to>
[07] |
[08] <to partnerLink="NCName" />
[09] |
[10] <to variable="BPELVariableName"
```

```
[11]     property="QName" />
[12] |
[13] <to expressionLanguage="anyURI"?>expression</to>
[14] |
[15] <to/>
```

Listing 24: BPEL Assign, to- Varianten, XML Struktur

5.4.4.3. Zusammenfassung

Eine BPEL Assign Aktivität wird auf eine WoG Regelmenge abgebildet, welche die Aktivierung und die Beendigung einer WoG Assign Aktivität modelliert. Bei der Aktivierung wird der Lebensmarker gelöscht und bei der Beendigung wird der Lebensmarker wieder erzeugt. Der allgemeine Aufbau eines Nichtterminals vom Aktivitätstyp Assign wird in Listing 25 dargestellt. Durch das Attribut „type“ in Zeile 1 wird der Typ des Nichtterminals festgelegt, in diesem Fall ist es ein Assign. Die Attribute „id“ (Zeile 2) und „name“ (Zeile 3) kommen bei allen Nichtterminalen vor und wurden bereits in Kapitel 5.4.1 beschrieben. Die Kindelemente (Zeile 4 bis 12) beschreiben die einzelnen Kopieroperationen und werden unverändert von BPEL übernommen.

```
[01] <nonTerminal xsi:type="tAssign"
[02]              id="NCName"
[03]              name="NCName"?>
[04]   (
[05]    <copy keepSrcElementName="yes|no"?
[06]          ignoreMissingFromData="yes|no"?>
[07]       from-spec to-spec
[08]    </copy>
[09]    |
[10]    <extensionAssignOperation>
[11]       assign-element-of-other-namespace
[12]    </extensionAssignOperation>  )+
[13] </nonTerminal>
```

Listing 25: WoG Assign, XML Struktur

Der Aufbau der Wog Regelmenge für ein BPEL Assign wird an einem Beispiel gezeigt. Listing 26 zeigt eine BPEL Assign Aktivität ASS1 mit zwei „copy“ Elementen, welche auf die Variable V1 schreibend und auf die Variable V2 lesend zugreifen. Regelmenge 30 modelliert diese BPEL Assign Aktivität ASS1 in WoG. Die in den Regeln verwendete WoG Assign Aktivität A_1 ist in Listing 27 dargestellt.

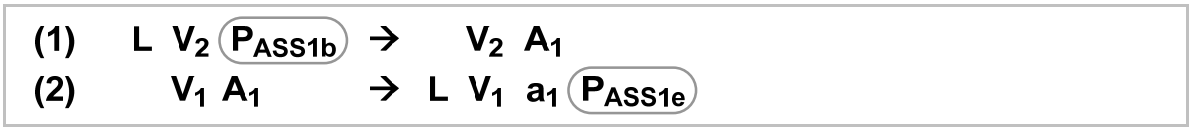
```
[01] <assign name="ASS1">
[02]    <copy>
[03]      <from>
[04]        <literal><![CDATA[<customer/>]]></literal>
[05]      </from>
[06]      <to variable="V1"/>
[07]    </copy>
[08]    <copy>
[09]     <from variable="V2"
[10]           part="customer_data"/>
[11]     <to variable="V1"/>
[12]    </copy>
[13] </assign>
```

Listing 26: BPEL Assign Aktivität ASS1

```
[01] <nonTerminal xsi:type="tAssign"
[02]              id="A1"
[03]              name="ASS1">
[04]   <copy>
[05]     <from>
[06]       <literal><![CDATA[<customer/>]]></literal>
[07]     </from>
[08]     <to variable="V1"/>
[09]   </copy>
[10]   <copy>
[11]    <from variable="V2"
[12]          part="customer_data"/>
[13]    <to variable="V1"/>
[14]   </copy>
[15] </nonTerminal>
```

Listing 27: WoG Assign Aktivität A_1

In Regel (1) der Regelmenge 30 aktiviert der Beginnplatzhalter P_{ASS1b} die WoG Assign Aktivität A_1. Bei dieser Aktivierung hat die Aktivität A_1 lesenden Zugriff auf die Variable V_2. Zusätzlich wird in dieser Regel der Lebensmarker L gelöscht. In Regel (2) wird die Aktivität A_1 beendet und sowohl der Lebensmarker als auch der Endeplatzhalter P_{ASS1e} werden erzeugt. Bei dieser Beendigung hat die Aktivität A_1 schreibenden Zugriff auf die Variable V_1.



Regelmenge 30: WoG Regelmenge für BPEL Assign

5.4.5. Throw

Eine BPEL Throw Aktivität erzeugt einen Fehler. Sie wird in WoG auf eine Regel abgebildet, die das entsprechende Fehlernichtterminal erzeugt. Im Kapitel 5.5.8.1 wird beschrieben, wie die BPEL Fehlerbehandlung in WoG modelliert wird.

5.4.6. Rethrow

Eine BPEL Rethrow Aktivität kann nur in BPEL Fault Handler verwendet werden. Sie gibt den aktuellen Fehler an den umgebenden Scope weiter. Die Modellierung von BPEL Fault Handler in WoG wird in Kapitel 5.5.8.1 beschrieben.

5.4.7. Wait

Eine BPEL Wait Aktivität beschreibt einen Wartevorgang, es kann eine Wartedauer angegeben werden oder ein Zeitpunkt, bis zu dem gewartet wird. Eine BPEL Wait Aktivität wird auf eine WoG Wait Aktivität abgebildet. Listing 28 zeigt den Aufbau einer BPEL Wait Aktivität. Die Kindelemente werden unverändert in die WoG Wait Aktivität übernommen. Der Aufbau einer WoG Wait Aktivität ist in Listing 29 dargestellt. Der in der BPEL Wait Aktivität verwendete Ausdruck (Zeile 4 und Zeile 6) wird bei der Übersetzung in WoG analysiert, um festzustellen, auf welche Variablen der Ausdruck zugreift. Diese Variablen werden in die Aktivierungsregel der entsprechenden WoG Wait Aktivität eingefügt.

```
[01] <wait standard-attributes>
[02]    standard-elements
[03]    (
[04]    <for expressionLanguage="anyURI"?>duration-expr</for>
[05]    |
[06]    <until expressionLanguage="anyURI"?>deadline-expr</until>
[07]    )
[08] </wait>
```

Listing 28: BPEL Wait, XML Struktur

```
[01] <nonTerminal xsi:type="tWait"
[02]              id="NCName"
[03]              name="NCName"?>
[04]    (
[05]    <for expressionLanguage="anyURI"?>duration-expr</for>
[06]    |
[07]    <until expressionLanguage="anyURI"?>deadline-expr</until>
[08]    )
[09] </nonTerminal>
```

Listing 29: WoG Wait, XML Struktur

5.4.8. Empty

Eine BPEL Empty Aktivität ist eine leere Aktivität, sie beschreibt keine Aktion. Eine BPEL Empty Aktivität wird auf eine WoG Regelmenge abgebildet. Listing 30 zeigt den Aufbau einer BPEL Empty Aktivität.

```
[01] <empty standard-attributes>
[02]    standard-elements
[03] </empty>
```

Listing 30: BPEL Empty, XML Struktur

In Regelmenge 31 wird die BPEL Empty Aktivität EMP1 in WoG modelliert. Der Beginnplatzhalter P_{EMP1b} geht direkt in den Endeplatzhalter P_{EMP1e} über.

(1) P_{EMP1b} → P_{EMP1e}

Regelmenge 31: WoG Regelmenge für BPEL Empty

5.4.9. ExtensionActivity

BPEL ist generell erweiterbar. Die Aktivität ExtensionActivity beschreibt explizit, dass BPEL um zusätzliche Aktivitätstypen erweiterbar ist. Die Übersetzung dieser Aktivität von BPEL auf WoG ist abhängig von den konkret verwendeten Erweiterungen und kann deswegen hier nicht näher beschrieben werden. Der Aufbau einer BPEL ExtensionActivity ist in Listing 31 dargestellt.

```
[01] <extensionActivity>
[02]    <anyElementQName standard-attributes>
[03]       standard-elements
[04]    </anyElementQName>
[05] </extensionActivity>
```

Listing 31: BPEL ExtensionActivity, XML Struktur

5.4.10. Exit

Eine BPEL Exit Aktivität beendet eine laufende Prozessinstanz. Alle laufenden Aktivitäten werden sofort beendet und es wird keine Fehlerbehandlung durchgeführt. Eine BPEL Exit Aktivität wird auf eine WoG Quit Aktivität abgebildet. Listing 32 zeigt den Aufbau einer BPEL Exit Aktivität.

```
[01] <exit standard-attributes>
[02]    standard-elements
[03] </exit>
```

Listing 32: BPEL Exit, XML Struktur

Regelmenge 32 modelliert die BPEL Exit Aktivität EX1 in WoG. In Regel (1) aktiviert der Beginnplatzhalter P_{EX1b} die WoG Quit Aktivität Q_1. Bei dieser Aktivierung wird der Lebensmarker L gelöscht und der Killmarker K erzeugt. Das Beenden einer Prozessinstanz mit der WoG Aktivität Quit wurde in Kapitel 3.2.10 ausführlich beschrieben. Die dort beschriebenen Regeln, die das Verhalten des Killmarkers K modellieren, werden hier nicht aufgeführt. Nachdem durch den Killmarker alle Aktivitäten beendet worden sind, wird in Regel (2) die Quit Aktivität Q_1 selbst beendet und in Regel (3) der Prozessscope gelöscht. Wichtig ist hierbei, dass der Platzhalter P_{EX1e} niemals erzeugt wird, da nach der Abarbeitung der WoG Regelmenge einer BPEL Exit Aktivität die gesamte Prozessinstanz beendet ist.

(1)	**L** P_{EX1b}	→	Q_1 **K**
	…	→	**…**
(2)	K_l Q_1 K_r	→	q_1
(3)	S_{PROCl} **K** S_{PROCr}	→	**ε**

Regelmenge 32: WoG Regelmenge für BPEL Exit

Der allgemeine Aufbau eines Nichtterminals vom Aktivitätstyp Quit wird in Listing 33 dargestellt. Durch das Attribut „type“ in Zeile 1 wird der Typ des Nichtterminals festgelegt, in diesem Fall ist es ein Quit. Die Attribute „id“ (Zeile 2) und „name“ (Zeile 3) kommen bei allen Nichtterminalen vor und wurden bereits in Kapitel 5.4.1 beschrieben.

```
[01] <nonTerminal xsi:type=“tQuit"
[02]              id="NCName"
[03]              name="NCName"?>
[04] </nonTerminal>
```

Listing 33: WoG Quit, XML Struktur

5.4.11. Compensate

Die BPEL Aktivität Compensate aktiviert die Compensation Handler der direkten Kindaktivitäten eine Scopes in der umgekehrten Ausführungsreihenfolge. Die Modellierung von Compensation wird in Kapitel 5.5.8.2 behandelt.

5.4.12. CompensateScope

Die BPEL Aktivität CompensateScope aktiviert den Compensation Handler eines bestimmten Scopes. Die Aktivierung von Compensation Handler wird in Kapitel 5.5.8.2 beschrieben.

5.4.13. Validate

Die BPEL Aktivität Validate validiert Variablen gegen ihr Schema. Sie wird in WoG auf eine WoG Assign Aktivität abgebildet, deren „validate“ Attribut auf „true“ gesetzt wird. Für jede zu validierende Variable wird ein „copy“ Element erstellt, welches die Variable auf sich selbst kopiert, also nichts tut. Somit werden alle angegebenen Variablen validiert.

5.5. BPEL Strukturierte Aktivitäten

Neben den bisher vorgestellten Basis Aktivitäten definiert BPEL weitere strukturierte Aktivitäten. Eine strukturierte Aktivität enthält eine oder mehre Aktivitäten und definiert die Reihenfolge, in der diese ausgeführt werden. Im Folgenden wird beschrieben, wie strukturierte BPEL Aktivitäten auf WoG abgebildet werden. Die Abbildung der Kindelemente selbst wird dabei nicht beschrieben. Dieses Vorgehen wurde bereits in Kapitel 5.1 vorgestellt. Die Regelmengen, welche die einzelnen BPEL Aktivitäten repräsentieren, werden durch Beginn- und Endeplatzhalter verknüpft. Das Erzeugen eines Beginnplatzhalters einer BPEL Aktivität aktiviert die entsprechende Regelmenge, sie kann jetzt abgearbeitet werden. Nach der Abarbeitung einer Regelmenge wird der Endeplatzhalter dieser BPEL Aktivität erzeugt.

5.5.1. Sequence

Eine BPEL Sequence Aktivität beschreibt die Hintereinanderausführung ihrer Kindaktivitäten. Listing 34 zeigt den Aufbau einer BPEL Sequence Aktivität. Zeile 3 besagt, dass sie ein oder mehrere Kindaktivitäten hat, die in der angegebenen Reihenfolge ausgeführt werden.

Eine BPEL Sequence Aktivität wird auf eine Menge von WoG Regeln abgebildet. In diesen Regeln werden die Beginn- und Endeplatzhalter der Kindaktivitäten verwendet, um ihre Hintereinanderausführung zu modellieren. Die Darstellung von sequentiellem Kontrollfluss in WoG wurde in Kapitel 3.2.3 vorgestellt. Dieses Konstrukt wird zur Modellierung einer BPEL Sequence Aktivität in WoG verwendet.

```
[01] <sequence standard-attributes>
[02]    standard-elements
[03]     activity+
[04] </sequence>
```

Listing 34: BPEL Sequence, XML Struktur

Listing 35 zeigt ein Beispiel für eine BPEL Sequence mit den Kindaktivitäten REC1, ASS1 und REP1. Dieses Beispiel wird durch Regelmenge 33 in WoG dargestellt.

```
[01] <sequence name="SEQ1" >
[02]   <receive name="REC1"
[03]            partnerLink="PL_1"
[04]            portType="ns1:PT_1"
[05]            operation="process"
[06]            variable="V1" >
[07]   </receive>
[08]   <assign name="ASS1" >
[09]     <copy>
[10]       <from variable="V1" />
[11]       <to variable="V2" />
[12]     </copy>
[13]   </assign>
[14]   <reply name="REP1"
[15]          partnerLink="PL_1"
[16]          portType="ns1:PT_1"
[17]          operation="process"
[18]          variable="V2" >
[19]   </reply>
[20] </sequence>
```

Listing 35: BPEL Sequence, Beispiel

In Regel (1) der Regelmenge 33 erzeugt der Beginnplatzhalter P_{SEQ1b} der Sequence SEQ1 den Beginnplatzhalter P_{REC1b} der ersten Kindaktivität der Sequence. Durch das Erzeugen dieses Beginnplatzhalters wird die entsprechende Regelmenge der Aktivität REC1 aktiviert. Sobald diese Regelmenge abgearbeitet wurde, wird der dazugehörige Endeplatzhalter P_{REC1e} erzeugt. Dieser wird in Regel (2) verwendet, um zu modellieren, dass nach dem Ende der Aktivität REC1 die Aktivität ASS1 aktiviert werden kann. Regel (3) ist nach dem gleichen Prinzip aufgebaut. Regel (4) besagt, dass nach dem Ende der letzten Aktivität in der Sequence, das Ende der gesamten Sequence erreicht worden ist.

(1)	P_{SEQ1b}	→	P_{REC1b}
(2)	P_{REC1e}	→	P_{ASS1b}
(3)	P_{ASS1e}	→	P_{REP1b}
(4)	P_{REP1e}	→	P_{SEQ1e}

Regelmenge 33: WoG Regelmenge für BPEL Sequence

5.5.2. If

Eine BPEL If Aktivität beschreibt bedingten Kontrollfluss abhängig von Booleschen Bedingungen. Sie wird auf eine Menge von WoG Regeln und eine oder mehrere WoG Evaluation Aktivitäten abgebildet. Die Modellierung von bedingtem Kontrollfluss in WoG wurde in den Kapiteln 3.2.9.1 und 3.3.1 vorgestellt. Diese Konstrukte werden zur Modellierung einer BPEL If Aktivität in WoG verwendet.

```
[01] <if standard-attributes>
[02]   standard-elements
[03]   <condition expressionLanguage="anyURI"?>bool-expr</condition>
[04]   activity
[05]   <elseif>*
[06]     <condition expressionLanguage="anyURI"?>bool-expr</condition>
[07]     activity
```

```
[08]   </elseif>
[09]   <else>?
[10]     activity
[11]   </else>
[12] </if>
```

Listing 36: BPEL If, XML Struktur

Listing 36 zeigt den Aufbau einer BPEL If Aktivität. Eine If Aktivität besteht aus einem if Zweig (Zeile 1 bis 4), optional beliebig vielen elseif Zweigen (Zeile 5 bis 8) und einem optionalen else Zweig (Zeile 9 bis 11). Zur Ausführungszeit werden die Bedingungen in der gegebenen Reihenfolge ausgewertet. Der erste Zweig, dessen Bedingung zu wahr ausgewertet wird, wird ausgeführt. Wenn keine Bedingung zu wahr ausgewertet wird, dann wird, falls vorhanden, der else Zweig ausgeführt. Jeder der Zweige einer If Aktivität enthält genau eine BPEL Aktivität.

```
[01] <if name="IF1" >
[02]   <condition>$V1 > 1000</condition>
[03]   <assign name="ASS1" >
[04]     <copy>
[05]       <from variable="V1" />
[06]       <to variable="V2" />
[07]     </copy>
[08]   </assign>
[09]   <elseif>
[10]     <condition>$V1 > 500</condition>
[11]     <assign name="ASS2" >
[12]       <copy>
[13]         <from variable="V1" />
[14]         <to variable="V3" />
[15]       </copy>
[16]     </assign>
[17]   </elseif>
[18]   <else>
[19]     <assign name="ASS3" >
[20]       <copy>
[21]         <from variable="V1" />
[22]         <to variable="V4" />
[23]       </copy>
[24]     </assign>
[25]   </else>
[26] </if>
```

Listing 37: BPEL „if then elseif else“, Beispiel

Listing 37 zeigt ein Beispiel für eine BPEL If Aktivität mit den Kindaktivitäten ASS1, ASS2 und ASS3. Dieses Beispiel wird durch Regelmenge 33 in WoG dargestellt. In Regel (1) aktiviert der Beginnplatzhalter P_{IF1b} der BPEL If Aktivität IF1 die WoG Evaluation Aktivität E_1. Diese Aktivität entspricht der Bedingung des if Zweiges der BPEL If Aktivität IF1. In Regel (2) wird E1 beendet. In Regel (3) und (4) wird das Ergebnis ausgewertet. Wird die Bedingung zu wahr ausgewertet, dann wird der Beginnplatzhalter P_{ASS1b} der Kindaktivität des if Zweiges erzeugt (Regel (3)). Andernfalls wird der Platzhalter P_{E2} der Bedingung des folgenden elseif Zweiges erzeugt (Regel (4)). In Regel (5) und (6) wird die Evaluation Aktivität E_2 aktiviert und beendet. Wird sie zu wahr ausgewertet, so wird der Beginnplatzhalter P_{ASS2b} der Kindaktivität des elseif Zweiges erzeugt (Regel (7)). Wird sie zu falsch ausgewertet, so wird, da es keine weiteren elseif Zweige gibt, der Beginnplatzhalter P_{ASS3b} der Kindaktivität des else Zweiges erzeugt (Regel (8)).

Die Nichtterminale T und F können nach dem Erzeugen gleich wieder gelöscht werden. Dies wird in den Regeln (9) und (10) dargestellt. Die Regeln (11) bis (13) beschreiben, dass die BPEL If Aktivität IF1 beendet ist, sobald eine ihrer Kindaktivitäten beendet ist. Bei der Ausführung einer BPEL If Aktivität wird höchstens eine ihrer Kindaktivitäten ausgeführt.

Besitzt eine BPEL IF Aktivität keinen else Zweig, so wird in WoG ihr Endeplatzhalter bereits in der entsprechenden Auswertungsregel der Evaluation Aktivität des letzten elseif Zweiges erzeugt. In dem gezeigten Beispiel wäre das die Regel (8), hier würde anstatt des Platzhalters P_{ASS3b} der Endeplatzhalter P_{IF1e} erzeugt werden.

(1)	$V_1 P_{IF1b}$	→	$V_1 E_1$
(2)	E_1	→	$e_1 R_{E1}$
(3)	R_{E1}	→	$T P_{ASS1b}$
(4)	R_{E1}	→	$F P_{E2}$
(5)	$V_1 P_{E2}$	→	$V_1 E_2$
(6)	E_2	→	$e_2 R_{E2}$
(7)	R_{E2}	→	$T P_{ASS2b}$
(8)	R_{E2}	→	$F P_{ASS3b}$
(9)	T	→	ε
(10)	F	→	ε
(11)	P_{ASS1e}	→	P_{IF1e}
(12)	P_{ASS2e}	→	P_{IF1e}
(13)	P_{ASS3e}	→	P_{IF1e}

Regelmenge 34: WoG Regelmenge für BPEL „if then elseif else“

Die in der Regelmenge 34 verwendeten WoG Evaluation Aktivitäten E1 und E2 sind in Listing 38 und Listing 39 dargestellt. Das Element „condition“ wird unverändert von der BPEL If Aktivität übernommen. Der allgemeine Aufbau einer WoG Evaluation Aktivität wird in Listing 40 gezeigt.

```
[01] <nonTerminal xsi:type="tEvaluation"
[02]              id="E1" >
[03]   <condition>$V1 > 1000</condition>
[04] </nonTerminal>
```

Listing 38: WoG Evaluation E1

```
[01] <nonTerminal xsi:type="tEvaluation"
[02]              id="E2" >
[03]   <condition>$V1 > 500</condition>
[04] </nonTerminal>
```

Listing 39: WoG Evaluation E2

```
[01] <nonTerminal xsi:type="tEvaluation"
[02]              id="NCName"
[03]              name="NCName"?>
[04]   <condition expressionLanguage="anyURI"?>bool-expr</condition>
[05] </nonTerminal>
```

Listing 40: WoG Evaluation, XML Struktur

5.5.3. While

Eine BPEL While Aktivität beschreibt eine Schleife, also die mehrmalige Hintereinanderausführung der gleichen Aktivität abhängig von einer Bedingung. Sie wird auf eine Menge von WoG Regeln und eine WoG Evaluation Aktivität abgebildet. Die Modellierung von Schleifen in WoG wurde in Kapitel 3.3.2 vorgestellt. Dieses Konstrukt wird zur Modellierung einer BPEL While Aktivität in WoG verwendet. Listing 41 zeigt den Aufbau einer BPEL While Aktivität. Sie besteht aus genau einer booleschen Bedingung und einer Kindaktivität.

```
[01] <while standard-attributes>
[02]   standard-elements
[03]   <condition expressionLanguage="anyURI"?>bool-expr</condition>
[04]   activity
[05] </while>
```

Listing 41: BPEL While, XML Struktur

Listing 42 zeigt ein Beispiel für eine BPEL While Aktivität mit der Kindaktivität SEQ1. Dieses Beispiel wird durch Regelmenge 35 in WoG dargestellt.

```
[01] <while name="WH1" >
[02]   <condition>$V1 > 10</condition>
[03]   <sequence name="SEQ1" >
[04]     <invoke name="INV1"
[05]             partnerLink="INV1_PL"
[06]             portType="ns1:BPEL_Invoke_01"
[07]             operation="notify" >
[08]     </invoke>
[09]     <assign name="ASS1" >
[10]       <copy>
[11]         <from>$V1 – 1</from>
[12]         <to variable="V1" />
[13]       </copy>
[14]     </assign>
[15]   </sequence>
[16] </while>
```

Listing 42: BPEL While, Beispiel

In Regel (1) der Regelmenge 35 aktiviert der Beginnplatzhalter P_{WH1b} der BPEL While Aktivität WH1 die Wog Evaluation Aktivität E_1. In Regel (2) wird sie beendet. In Regel (3) und (5) wird das Ergebnis ausgewertet. Ist die Bedingung wahr, dann wird der Beginnplatzhalter P_{SEQ1b} der Kindaktivität der BPEL While Aktivität erzeugt (Regel (3)). Ist die Bedingung falsch, dann wird der Endeplatzhalter P_{WH1e} der BPEL While Aktivität erzeugt (Regel (5)). In Regel (4) ist modelliert, dass nach dem Beenden der Kindaktivität die Schleifenbedingung erneut ausgewertet wird.

(1) V_1 (P_{WH1b}) → V_1 E_1
(2) E_1 → e_1 R_{E1}
(3) R_{E1} → T P_{SEQ1b}
(4) V_1 P_{SEQ1e} → V_1 E_1
(5) R_{E1} → F (P_{WH1e})

Regelmenge 35: WoG Regelmenge für BPEL While

5.5.4. RepeatUntil

Eine BPEL RepeatUntil Aktivität beschreibt eine Schleife, in der die Bedingung am Ende eines Durchlaufs ausgewertet wird. Sie wird also mindestens einmal ausgeführt. Die Abbildung auf WoG erfolgt ähnlich wie die Abbildung einer BPEL While Aktivität. Listing 43 zeigt den Aufbau einer BPEL RepeatUntil Aktivität. Sie besteht aus genau einer booleschen Bedingung und einer Kindaktivität.

```
[01] <repeatUntil standard-attributes>
[02]   standard-elements
[03]   activity
[04]   <condition expressionLanguage="anyURI"?>bool-expr</condition>
[05] </repeatUntil>
```

Listing 43: BPEL RepeatUntil, XML Struktur

Listing 44 zeigt ein Beispiel für eine BPEL RepeatUntil Aktivität mit der Kindaktivität SEQ1. Dieses Beispiel wird durch Regelmenge 36 in WoG dargestellt. In Regel (1) wird zuerst die Kindaktivität SEQ1 aktiviert. Erst nachdem sie beendet wurde, wird in Regel (2) und (3) die Schleifenbedingung ausgewertet. Anschließend kann die Schleife entweder mit Regel (4) erneut ausgeführt oder mit Regel (5) beendet werden.

```
[01] <repeatUntil name="RPU1" >
[02]   <sequence name="SEQ1" >
[03]     <invoke name="INV1"
[04]             partnerLink="INV1_PL"
[05]             portType="ns1:BPEL_Invoke_01"
[06]             operation="notify" >
[07]     </invoke>
[08]     <assign name="ASS1" >
[09]       <copy>
[10]         <from>$V1 – 1</from>
[11]         <to variable="V1" />
[12]       </copy>
[13]     </assign>
[14]   </sequence>
[15]   <condition>$V1 > 10</condition>
[16] </repeatUntil>
```

Listing 44: BPEL RepeatUntil Aktivität RPU1

(1)	P_{RPU1b}	→	P_{SEQ1b}
(2)	$V_1\ P_{SEQ1e}$	→	$V_1\ E_1$
(3)	E_1	→	$e_1\ R_{E1}$
(4)	R_{E1}	→	$T\ P_{SEQ1b}$
(5)	R_{E1}	→	$F\ P_{RPU1e}$

Regelmenge 36: WoG Regelmenge für BPEL RepeatUntil

5.5.5. Pick

Eine BPEL Pick Aktivität beschreibt das Empfangen einer Nachricht über eine von mehreren möglichen Web Service Schnittstellen. Es ist ebenfalls möglich, dass sie auf Grund einer zeitlichen Bedingung beendet wird, ohne eine Nachricht zu empfangen. Eine BPEL Pick Aktivität wird abgebildet auf eine WoG Regelmenge sowie auf eine WoG In Aktivität und gegebenenfalls eine oder mehrere WoG Wait Aktivitäten.

```
[01] <pick createInstance="yes|no"?
[02]       standard-attributes>
[03]   standard-elements
[04]   <onMessage partnerLink="NCName"
[05]              portType="QName"?
[06]              operation="NCName"
[07]              variable="BPELVariableName"?
[08]              messageExchange="NCName"?>+
[09]     <correlations>?
[10]       <correlation set="NCName"
[11]                    initiate="yes|join|no"? />+
[12]     </correlations>
[13]     <fromParts>?
[14]       <fromPart part="NCName"
[15]                 toVariable="BPELVariableName" />+
[16]     </fromParts>
[17]     activity
[18]   </onMessage>
[19]   <onAlarm>*
[20]     (
[21]       <for expressionLanguage="anyURI"?>duration-expr</for>
[22]       |
[23]       <until expressionLanguage="anyURI"?>deadline-expr</until>
[24]     )
[25]     activity
[26]   </onAlarm>
[27] </pick>
```

Listing 45: BPEL Pick, XML Struktur

Listing 45 zeigt den Aufbau einer BPEL Pick Aktivität. Sie enthält mindestens ein „onMessage“ Element und kann zusätzlich ein oder mehrere „onAlarm“ Elemente enthalten.

Für eine BPEL Pick Aktivität wird in WoG genau eine In Aktivität erzeugt. Für jedes „onMessage“ Element in der BPEL Pick Aktivität wird ein entsprechendes „interface“ Element in der dazugehörigen WoG In Aktivität erzeugt. In Kapitel 5.4.2 wurde die Abbildung einer BPEL Receive Aktivität auf WoG beschrieben. Da der Aufbau eines „onMessage“ Elementes im Wesentlichen

dem Aufbau einer BPEL Receive Aktivität entspricht, wird die Abbildung eines „onMessage“ Elementes auf WoG hier nicht näher beschrieben.

Für jedes „onAlarm“ Element einer BPEL Pick Aktivität wird eine WoG Wait Aktivität erzeugt. In Kapitel 5.4.7 wurde die Abbildung einer BPEL Wait Aktivität auf WoG beschrieben. Da der Aufbau eines „onAlarm“ Elementes im Wesentlichen dem Aufbau einer BPEL Wait Aktivität entspricht, wird die Abbildung eines „onAlarm“ Elementes auf WoG hier nicht näher beschrieben.

Die in einer BPEL Pick Aktivität enthaltenen „onMessage“ und „onAlarm“ Elemente werden bei der Aktivierung der Pick Aktivität alle gleichzeitig aktiv. Das bedeutet, dass in WoG die entsprechende WoG In Aktivität sowie die dazugehörigen WoG Wait Aktivitäten ebenfalls gleichzeitig aktiviert werden müssen. Dies ist nur dann möglich, wenn diese Aktivitäten in der gleichen Regel aktiviert werden.

Listing 46 zeigt ein Beispiel für eine BPEL Pick Aktivität PIC1 mit zwei „onMessage“ Elementen und einem „onAlarm“ Element. Die Pick Aktivität wartet entweder auf eine Bestellbestätigung oder auf eine Ablehnung der Bestellung. Wird über einen Zeitraum von 3 Tagen und 10 Stunden keines von beiden empfangen, so wird die Pick Aktivität beendet. Dieses Beispiel wird durch Regelmenge 37 in WoG dargestellt.

```
[01] <pick name="PIC1">
[02]    <onMessage partnerLink="PIC1_PL1"
[03]               portType="ns1:PT_01"
[04]               operation="acceptOrder"
[05]               variable="V1">
[06]    </onMessage>
[07]    <onMessage partnerLink="PIC1_PL2"
[08]               portType="ns1:PT_02"
[09]               operation="rejectOrder"
[10]               variable="V2">
[11]    </onMessage>
[12]    <onAlarm>
[13]       <for>’P3DT10H’</for>
[14]    </onAlarm>
[15] </pick>
```

Listing 46: BPEL Pick, Beispiel

In Regel (1) der Regelmenge 37 aktiviert der Beginnplatzhalter P_{PIC1b} der Pick Aktivität PIC1 gleichzeitig die WoG In Aktivität I_1 und die WoG Wait Aktivität W_1. Die Beendigung dieser Aktivitäten ist ebenfalls nur gleichzeitig möglich. Dies ist durch die Regeln (2) und (3) dargestellt.

Regel (2) betrachtet den Fall, dass die Beendigung durch die WoG Wait Aktivität W_1 ausgelöst worden ist. Die WoG In Aktivität I_1 wird in diesem Fall vorzeitig beendet, ohne dass sie eine Nachricht empfangen hat. Da die BPEL Pick Aktivität damit abgearbeitet worden ist, wird in Regel (2) zusätzlich ihr Endeplatzhalter P_{PIC1e} erzeugt.

Regel (3) betrachtet den Fall, dass die Beendigung durch die WoG In Aktivität I_1 ausgelöst worden ist. Die WoG Wait Aktivität W_1 wird in diesem Fall vorzeitig beendet. Die empfangene Nachricht wird zunächst in der Ergebnisvariable R_{I1} zwischengespeichert. Regel (4) und (5) beschreiben, dass die Nachricht, abhängig von der Schnittstelle, über die sie empfangen wurde, in die passende Variable abgespeichert wird. Da die BPEL Pick Aktivität damit abgearbeitet worden ist, wird in beiden Regeln zusätzlich ihr Endeplatzhalter P_{PIC1e} erzeugt.

(1) P_{PIC1b} → I_1 W_1
(2) I_1 W_1 → i_1 w_1 P_{PIC1e}
(3) I_1 W_1 → i_1 w_1 R_{I1}
(4) V_1 R_{I1} → V_1 M_1 P_{PIC1e}
(5) V_2 R_{I1} → V_2 M_2 P_{PIC1e}
(6) M_1 → ε
(7) M_2 → ε

Regelmenge 37: WoG Regelmenge für BPEL Pick

Die in der Regelmenge 37 verwendeten WoG Aktivitäten I_1 und W_1 werden in Listing 47 und Listing 48 dargestellt. Die Nichtterminale M_1 und M_2 beschreiben die einzelnen Schnittstellen, über welche Nachrichten empfangen werden können und ermöglichen so zur Laufzeit die Auswahl der passenden Regel. M_1 und M_2 werden in Listing 49 und Listing 50 beschrieben. Ihre allgemeine Struktur ist in Listing 51 dargestellt.

```
[01] <nonTerminal xsi:type="tIn"
[02]              id="I1"
[03]              name="PIC1">
[04]   <interface partnerLink="PIC1_PL1"
[05]              portType="ns1:PT_01"
[06]              operation="acceptOrder >
[07]   </interface>
[08]   <interface partnerLink="PIC1_PL2"
[09]              portType="ns1:PT_02"
[10]              operation="rejectOrder" >
[11]    </interface>
[12] </nonTerminal>
```

Listing 47: WoG In I_1

```
[01] <nonTerminal xsi:type="tWait"
[02]              id="W1"
[03]              name="PIC1">
[04]   <for>’P3DT10H’</for>
[05] </nonTerminal>
```

Listing 48: WoG Wait W_1

```
[01] <nonTerminal xsi:type="tMessageInterface"
[02]              id="M1"
[03]              name="PIC1">
[04]   <interface partnerLink="PIC1_PL1"
[05]              portType="ns1:PT_01"
[06]              operation="acceptOrder" >
[07]   </interface>
[08] </nonTerminal>
```

Listing 49: WoG MessageInterface M_1

```
[01] <nonTerminal xsi:type="tMessageInterface"
[02]              id="M2"
[03]              name="PIC1">
[04]   <interface partnerLink="PIC1_PL2"
[05]              portType="ns1:PT_02"
[06]              operation="rejectOrder" >
[07]    </interface>
[08] </nonTerminal>
```

Listing 50: WoG MessageInterface M_2

```
[01] <nonTerminal xsi:type="tMessageInterface"
[02]              id="NCName"
[03]              name="NCName"?
[04]   <interface partnerLink="NCName"
[05]              portType="QName"?
[06]              operation="NCName"
[07]              messageExchange="NCName"? >
[08]     <correlations>?
[09]       <correlation set="NCName"
[10]                    initiate="yes|join|no"? />+
[11]     </correlations>
[12]   </interface>
[13] </nonTerminal>
```

Listing 51: WoG MessageInterface, XML Struktur

5.5.6. Flow

BPEL Prozesse basieren auf einer Blockstruktur. Strukturierte Aktivitäten können andere Aktivitäten enthalten. Zusätzlich ist es aber auch möglich, Teile eines Prozesses graphbasiert zu modellieren. Dies geschieht mit Hilfe der Flow Aktivität. Eine BPEL Flow Aktivität wird abgebildet auf einen WoG Scope mit darin enthaltenen WoG Assign und Evaluation Aktivitäten sowie zusätzlichen Variablen.

Listing 52 zeigt den Aufbau einer BPEL Flow Aktivität. Sie enthält ein oder mehrere Kindaktivitäten (Zeile 6). Innerhalb einer Flow Aktivität können außerdem „Links“ definiert werden (Zeile 3 bis 5). Ein Link repräsentiert den bedingten Kontrollfluss zwischen zwei Aktivitäten innerhalb eines Flows. Die Definition der entsprechenden Bedingung sowie die Verknüpfung eines Links mit seiner Quell- und Zielaktivität geschehen in diesen Aktivitäten und nicht bei der Definition des Links selbst.

Links können nicht nur die direkten Kinder eines Flows verbinden, sie können, mit einigen Einschränkungen, auch indirekte Kindaktivitäten (also Kinder von Kindern) eines Flows als Quell- oder Zielaktivität haben. Links, welche die Grenzen von strukturierten BPEL Aktivitäten überqueren, nennt man auch „Cross Boundary Links“. Bei der Übersetzung von BPEL auf WoG werden Cross Boundary Links nicht betrachtet.

```
[01] <flow standard-attributes>
[02]   standard-elements
[03]   <links>?
[04]     <link name="NCName">+
[05]   </links>
[06]   activity+
[07] </flow>
```

Listing 52: BPEL Flow, XML Struktur

Listing 53 zeigt die Attribute und Elemente, die in jeder Aktivität innerhalb eines Flows verwendet werden können, um graphbasierten Kontrollfluss zu modellieren. Eine Aktivität kann Ausgangspunkt von einem oder mehreren Links sein. Dies ist in Zeile 9 bis 15 dargestellt. Zu jedem Link kann eine Boolesche Bedingung, die sogenannte „Transition Condition“ angegeben werden (Zeile 11 bis 13). Nachdem eine Aktivität beendet worden ist, werden die Bedingungen aller von ihr ausgehenden Links ausgewertet und die Ergebnisse gespeichert. Ist zu einem Link keine Bedingung angegeben, dann ergibt die Auswertung immer „wahr“.

Eine Aktivität kann Endpunkt von einem oder mehreren Links sein. Dies ist in Zeile 3 bis 8 dargestellt. Ist eine Aktivität Ziel von mindestens einem Link, so kann in dieser Aktivität eine Boolesche Bedingung, die sogenannte „Join Condition“ angegeben werden (Zeile 4 bis 6). Die Auswertung der Join Condition bestimmt zur Laufzeit, ob diese Aktivität aktiviert wird oder nicht. Die Join Condition wird erst dann ausgewertet, wenn die Transition Conditions aller eingehenden Links ausgewertet worden sind. Die Join Condition bezieht sich ausschließlich auf die Auswertungsergebnisse der Transition Conditions aller eingehenden Links. Ist keine Join Condition angegeben, dann wird eine implizite Join Condition angenommen. Diese implizite Join Condition ist immer eine ODER Verknüpfung der Auswertungsergebnisse der Transition Conditions aller eingehenden Links.

Wird zur Laufzeit die Join Condition einer Aktivität zu „falsch“ ausgewertet, dann wird diese Aktivität nicht aktiviert. Das Attribut „suppressJoinFailure“ bestimmt, was anschließend geschieht (Zeile 2). Ist dieses Attribut auf „no“ gesetzt, dann wird ein Fehler vom Typ „bpel:joinFailure“ erzeugt. Ist dieses Attribut auf „yes“ gesetzt, dann wird kein Fehler erzeugt. Stattdessen werden die Transition Conditions aller von dieser Aktivität ausgehenden Links zu „falsch“ ausgewertet. Dieses Verfahren wird „Dead Path Elimination“ genannt.

```
[01] <anyBPELActivity name="NCName"?
[02]                  suppressJoinFailure="yes|no"?>
[03]   <targets>?
[04]     <joinCondition expressionLanguage="anyURI"?>?
[05]        bool-expr
[06]     </joinCondition>
[07]     <target linkName="NCName" />+
[08]   </targets>
[09]   <sources>?
[10]     <source linkName="NCName">+
[11]       <transitionCondition expressionLanguage="anyURI"?>?
[12]         bool-expr
[13]       </transitionCondition>
[14]     </source>
[15]   </sources>
[16] </anyBPELActivity>
```

Listing 53: BPEL Standardattribute und Standardelemente, XML Struktur

Die Transformation einer BPEL Flow Aktivität nach WoG wird anhand eines Beispiels beschrieben. Abbildung 38 zeigt eine BPEL Flow Aktivität FLO1 in Form eines Graphen. Die Spezifikation dieser BPEL Flow Aktivität ist in Listing 54 dargestellt. Die Modellierung dieser Aktivität in WoG zeigt Regelmenge 38.

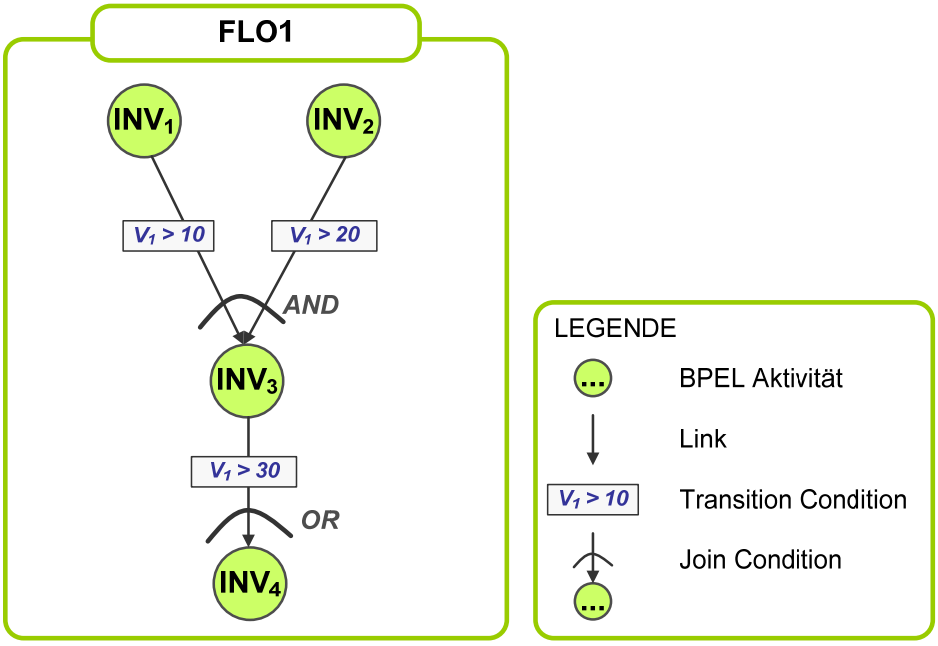


Abbildung 38: BPEL Flow Aktivität FLO1

```
[01] <flow name="FLO1" >
[02]   <links>
[03]     <link name="L1" />
[04]     <link name="L2" />
[05]     <link name="L3" />
[06]   </links>
[07]   <invoke name="INV1" …>
[08]     <sources>
[09]       <source linkName="L1" >
[10]          <transitionCondition>$V1 > 10</transitionCondition>
[11]       </source>
[12]     </sources>
[13]   </invoke>
[14]   <invoke name="INV2" …>
[15]     <sources>
[16]       <source linkName="L2" >
[17]          <transitionCondition>$V1 > 20</transitionCondition>
[18]       </source>
[19]     </sources>
[20]   </invoke>
[21]   <invoke name="INV3"
[22]           suppressJoinFailure="yes"  …>
[23]     <targets>
[24]       <joinCondition>$L1 and $L2</joinCondition>
[25]       <target linkName="L1" />
[26]       <target linkName="L2" />
[27]     </targets>
[28]     <sources>
[29]       <source linkName="L3" >
[30]          <transitionCondition>$V1 > 30</transitionCondition>
[31]       </source>
[32]     </sources>
[33]   </invoke>
[34]   <invoke name="INV4" …>
[35]     <targets>
```

```
[36]       <target linkName="L3" />
[37]     </targets>
[38]   </invoke>
[39] </flow>
```

Listing 54: BPEL Flow Aktivität FLO1

In Regel (1) der Regelmenge 38 erzeugt der Beginnplatzhalter P_{FLO1b} einen Scope S_1 mit Variablen und Platzhaltern. Für jeden Link L_i der BPEL Flow Aktivität FLO1 wird eine Variable V_{Li} vom Datentyp Boolean erzeugt und mit „falsch“ initialisiert. Diese Variable wird in WoG verwendet, um das Ergebnis der Auswertung der BPEL Transition Condition zu speichern. Die im Beispiel verwendete WoG Variable V_{L1} wird in Listing 58 dargestellt. Die Initialisierung der Variable mit dem Wert „falsch“ wird in den Zeilen 5 bis 7 beschrieben. Der allgemeine Aufbau einer WoG Variablen ist in Listing 59 dargestellt. Der Datentyp kann, ebenso wie in BPEL, über die Attribute „type“, „messageType“ oder „element“ angegeben werden. Die optionale Initialisierung erfolgt über ein „from“ Element. Dieses wurde bereits in Kapitel 5.4.4.1 näher beschrieben.

Der Kontrollfluss einer BPEL Flow Aktivität beginnt bei allen Kindaktivitäten, die nicht Ziel eines Links sind. Im dargestellten Beispiel sind es die BPEL Invoke Aktivitäten INV1 und INV2. Dementsprechend werden in Regel (1) ihre Beginnplatzhalter P_{INV1b} und P_{INV2b} erzeugt.

Die Auswertung einer Transition Condition wird in WoG durch eine WoG Assign Aktivität dargestellt. Zu jedem Link L_i wird eine WoG Assign Aktivität A_{Li} erstellt, welche die Auswertung der entsprechenden Transition Condition in die zu diesem Link gehörende WoG Variable V_{Li} kopiert. Die zu den BPEL Links L_1 und L_2 gehörenden WoG Assign Aktivitäten A_{L1} und A_{L2} sind in Listing 55 und Listing 56 dargestellt.

Regel (2) zeigt, dass nach der Ausführung der Regelmenge, welche die BPEL Invoke Aktivität INV1 repräsentiert, die Transition Condition des ausgehenden Links L_1 durch die WoG Assign Aktivität A_{L1} ausgewertet wird. Regel (3) zeigt, dass bei Beendigung der Aktivität A_{L1} das Ergebnis der Auswertung in der WoG Variablen V_{L1} abgespeichert wird und ein Platzhalter P_{E_INV3m} erzeugt wird. Regel (4) und (5) modellieren das gleiche Verhalten für die Aktivität INV2 und den Link L_2.

Die Join Condition der BPEL Aktivität INV3 wird in WoG auf die Evaluation Aktivität E_{INV3} abgebildet, diese ist in Listing 57 dargestellt. Die WoG Aktivität E_{INV3} darf erst dann aktiviert werden, wenn die Transition Conditions aller eingehenden Links von INV3 ausgewertet worden sind, d.h. in WoG, wenn die WoG Assign Aktivitäten A_{L1} und A_{L2} beendet worden sind. Die in Regel (3) und (5) erzeugten Platzhalter P_{E_INV3m} und P_{E_INV3} stellen dies sicher. In Regel (6) ist dargestellt, dass die WoG Aktivität E_{INV3} erst dann aktiviert werden kann, wenn beide Platzhalter erzeugt worden sind. In Regel (7) wird E_{INV3} beendet, Regel (8) und (11) stellen die Auswertung des Ergebnisses dar.

Wird die Join Condition der BPEL Aktivität INV3 zu „wahr“ ausgewertet, dann wird diese Aktivität aktiviert. Dies ist in Regel (8) dargestellt. Regel (9) und (10) stellen dar, dass nach dem Beenden von INV3 die Transition Condition des Links L_3 ausgewertet und abgespeichert wird. Zudem wird der Platzhalter P_{E_INV4} für die Auswertung der Join Condition der Folgeaktivität INV4 erzeugt.

Wird die Join Condition der BPEL Aktivität INV3 zu „falsch“ ausgewertet, dann wird im hier gezeigten Beispiel Dead Path Elimination ausgeführt (Listing 54, Zeile 22). Die Aktivität INV3 wird nicht ausgeführt, es wird direkt der Platzhalter P_{E_INV4} für die Auswertung der Join Condition der Folgeaktivität INV4 erzeugt. Der von INV3 ausgehende Link L_3 muss nicht explizit zu „falsch“ ausgewertet werden, da alle zu einem Link gehörenden Variablen immer mit „falsch“ initialisiert werden. In BPEL findet Dead Path Elimination zur Laufzeit statt. In WoG wird dieses Verhalten bereits in den Regeln explizit modelliert.

Die Regeln (12) und (13) beschreiben die Auswertung der Join Condition von INV4. Obwohl in diesem Beispiel keine explizite Join Condition für die BPEL Aktivität INV4 angegeben worden ist (Listing 54, Zeile 35 bis 37), wird in WoG eine WoG Evaluation Aktivität E_{INV4} erzeugt. Sie repräsentiert in diesem Fall die implizite BPEL Default Join Condition, eine ODER Verknüpfung der Auswertung aller eingehenden Links von INV4. Wird die Join Condition von INV4 zu „wahr" ausgewertet, dann wird der Beginnplatzhalter P_{INV4b} erzeugt (Regel (14)). Regel (15) modelliert, das INV4 keine ausgehenden Links hat, der Kontrollfluss endet nach der Beendigung von INV4. Wird die Join Condition von INV4 zu „falsch" ausgewertet, so endet der Kontrollfluss bereits nach der Auswertung der Join Condition von INV4 (Regel (16)). In Regel (19) wird der zum BPEL Flow FLO1 gehörende WoG Scope und alle darin lebenden Variablen gelöscht und der Endeplatzhalter P_{FLO1e} erzeugt.

(1) $P_{FLO1b} \rightarrow S_{1l}\ V_{L1}\ V_{L2}\ V_{L3}\ P_{INV1b}\ P_{INV2b}\ S_{1r}$
(2) $V_1\ P_{INV1e} \rightarrow V_1\ A_{L1}$
(3) $V_{L1}\ A_{L1} \rightarrow V_{L1}\ a_{L1}\ P_{E\_INV3m}$
(4) $V_1\ P_{INV2e} \rightarrow V_1\ A_{L2}$
(5) $V_{L2}\ A_{L2} \rightarrow V_{L2}\ a_{L2}\ P_{E\_INV3}$
(6) $V_{L1}\ V_{L2}\ P_{E\_INV3m}\ P_{E\_INV3} \rightarrow V_{L1}\ V_{L2}\ E_{INV3}$
(7) $E_{INV3} \rightarrow e_{INV3}\ R_{E\_INV3}$
(8) $R_{E\_INV3} \rightarrow T\ P_{INV3b}$
(9) $V_1\ P_{INV3e} \rightarrow V_1\ A_{L3}$
(10) $V_{L3}\ A_{L3} \rightarrow V_{L3}\ a_{L3}\ P_{E\_INV4}$
(11) $R_{E\_INV3} \rightarrow F\ P_{E\_INV4}$
(12) $V_{L3}\ P_{E\_INV4} \rightarrow V_{L3}\ E_{INV4}$
(13) $E_{INV4} \rightarrow e_{INV4}\ R_{E\_INV4}$
(14) $R_{E\_INV4} \rightarrow T\ P_{INV4b}$
(15) $P_{INV4e} \rightarrow \varepsilon$
(16) $R_{E\_INV4} \rightarrow F$
(17) $T \rightarrow \varepsilon$
(18) $F \rightarrow \varepsilon$
(19) $S_{1l}\ V_{L1}\ V_{L2}\ V_{L3}\ S_{1r} \rightarrow P_{FLO1e}$

Regelmenge 38: WoG Regelmenge für BPEL Flow

```
[01] <nonTerminal xsi:type="tAssign"
[02]              id="A_L1">
[03]   <copy>
[04]     <from>
[05]       <expression>$V1 > 10</expression>
[06]     </from>
[07]     <to variable="V_L1"/>
[08]   </copy>
[09] </nonTerminal>
```

Listing 55: WoG Assign A_{L1}

```
[01] <nonTerminal xsi:type="tAssign"
[02]              id="A_L2">
[03]   <copy>
[04]     <from>
[05]       <expression>$V1 > 20</expression>
[06]     </from>
[07]     <to variable="V_L2"/>
[08]   </copy>
[09] </nonTerminal>
```

Listing 56: WoG Assign A_{L2}

```
[01] <nonTerminal xsi:type="tEvaluation"
[02]              id="E_INV3" >
[03]   <condition>$V_L1 and $V_L2</condition>
[04] </nonTerminal>
```

Listing 57: WoG Evaluation E_{INV3}

```
[01] <nonTerminal xsi:type="tVariable"
[02]              id="V_L1"
[03]              variableName="L1"
[04]              type="xsd:boolean">
[05]   <from>
[06]       <literal>false</literal>
[07]   </from>
[08] </nonTerminal>
```

Listing 58: WoG Variable V_{L1}

```
[01] <nonTerminal xsi:type="tVariable"
[02]              id="NCName"
[03]              variableName="WOGVariableName"
[04]              messageType="QName"?
[05]              type="QName"?
[06]              element="QName"?>
[07]   from-spec?
[08] </nonTerminal>
```

Listing 59: WoG Variable, XML Struktur

5.5.6.1. Zusammenfassung

Bei der Abbildung einer BPEL Flow Aktivität auf WoG wird zunächst ein WoG Scope erzeugt. Dabei wird für jeden Link des BPEL Flows eine Boolesche Variable erzeugt, welche die Auswertung der Transition Condition des Links repräsentiert. Zusätzlich wird für jeden Link eine WoG Assign Aktivität erzeugt. Diese wertet die implizite oder explizite Transition Condition aus und speichert das Ergebnis. Für jede Kindaktivität des BPEL Flows, die mindestens einen eingehenden Link hat, wird eine WoG Evaluation Aktivität erzeugt. Diese repräsentiert die Auswertung der impliziten oder expliziten Join Condition der entsprechenden BPEL Aktivität. Dead Path Elimination wird in WoG explizit in den Regeln modelliert.

5.5.7. ForEach

Eine BPEL ForEach Aktivität beschreibt eine Schleife, bei der die Anzahl der Durchläufe über einen Zähler bestimmt wird. Die einzelnen Durchläufe einer BPEL ForEach Schleife können

sequentiell oder parallel ausgeführt werden. Die Abbildung auf WoG ist abhängig davon, ob die Schleife sequentiell oder parallel ausgeführt werden soll.

```
[01] <forEach counterName="BPELVariableName"
[02]          parallel="yes|no"
[03]          standard-attributes>
[04]    standard-elements
[05]    <startCounterValue expressionLanguage="anyURI"?>
[06]       unsigned-integer-expression
[07]    </startCounterValue>
[08]    <finalCounterValue expressionLanguage="anyURI"?>
[09]       unsigned-integer-expression
[10]    </finalCounterValue>
[11]    <completionCondition>?
[12]       <branches expressionLanguage="anyURI"?
[13]                 successfulBranchesOnly="yes|no"?>?
[14]                 unsigned-integer-expression
[15]       </branches>
[16]    </completionCondition>
[17]    <scope …>…</scope>
[18] </forEach>
```

Listing 60: BPEL ForEach, XML Struktur

Listing 60 zeigt den Aufbau einer BPEL ForEach Aktivität. Sie besitzt genau eine Kindaktivität vom Typ Scope (Zeile 17). Über das Attribut „parallel“ in Zeile 2 wird angegeben, ob die einzelnen Schleifendurchläufe sequentiell oder parallel ausgeführt werden sollen. In jedem Durchlauf der Schleife wird eine spezielle Variable deklariert. Der Name der Variable wird über das Attribut „counterName“ angegeben. Der Startwert der Variablen im ersten Durchlauf wird über das Element „startCounterValue“ definiert. In jedem Schleifendurchlauf wird der Wert der Variablen um eins erhöht, bis sie im letzten Durchlauf den durch das Element „finalCounterValue“ definierten Endwert hat. Die Anzahl der Schleifendurchläufe wird also berechnet durch (finalCounterValue – startCounterValue) + 1.

Die Ausführung einer ForEach Schleife kann vorzeitig abgebrochen werden. Die entsprechende Abbruchbedingung wird in den Zeilen 11 bis 16 definiert. Sie gibt an, wie viele Schleifendurchläufe beendet sein müssen, bevor die Schleife vorzeitig beendet wird. Über das Attribut „successfulBranchesOnly“ kann angegeben werden, ob dabei nur die erfolgreichen Durchläufe oder alle Durchläufe gezählt werden. Im Folgenden wird einmal für den sequentiellen und einmal für den parallelen Schleifendurchlauf beschrieben, wie die BPEL ForEach Aktivität auf WoG abgebildet wird.

5.5.7.1. Serieller Schleifendurchlauf

Listing 61 zeigt ein Beispiel einer ForEach Schleife FOR1, die seriell ausgeführt werden soll. Die Schleifenvariable „counter“ läuft von 13 bis 15, die Schleife wird also dreimal durchlaufen. Die Abbruchbedingung gibt allerdings an, dass die ForEach Aktivität bereits nach zwei Schleifendurchläufen beendet werden soll. In jeden Schleifendurchlauf wird der Schleifenkörper SCO1 ausgeführt.

```
[01] <forEach name="FOR1"
[02]          counterName="counter"
[03]          parallel="no">
[04]    <startCounterValue>13</startCounterValue>
[05]    <finalCounterValue>15</finalCounterValue>
[06]    <completionCondition>
```

```
[07]       <branches>2</branches>
[08]    </completionCondition>
[09]    <scope name="SCO1">…</scope>
[10] </forEach>
```

Listing 61: BPEL ForEach FOR1

Regelmenge 39 zeigt die BPEL ForEach Aktivität FOR1 in WoG. In Regel (1) erzeugt der Beginnplatzhalter P_{FOR1b} einen Scope mit zwei Variablen und einen Platzhalter. Die Variable $V_{WoG\_counter}$ ist eine in WoG zusätzlich erzeugte Zählvariable der Schleife. Sie wird mit dem Startwert (BPEL „startCounterValue“) initialisiert, im Beispiel also mit dem Wert 13. Ihr Wert wird in jedem Schleifendurchlauf um eins erhöht und ansonsten nicht verändert. Die Variable $V_{counter}$ entspricht der in BPEL angegebenen Schleifenvariable. Sie erhält zu Beginn jedes Schleifendurchlaufs den Wert der Variablen $V_{WoG\_counter}$, im Gegensatz zu dieser kann sie während eines Schleifendurchlaufs verändert werden.

Zu Beginn jeden Schleifendurchlaufs wird mit Regel (2) und (3) die Evaluation Aktivität E_1 ausgeführt. Sie ist in Listing 62 dargestellt. Die dort in Zeile 3 und 4 angegebene Bedingung kombiniert einerseits ob der Schleifenzähler den Endwert (BPEL „finalCounterValue“) überschritten hat und andererseits, ob die Abbruchbedingung (BPEL „completionCondition“) erfüllt ist. Ist eines von beiden der Fall, so wird Regel (9) angewendet. Anschließend kann mit Regel (12) der Scope S_1 gelöscht und der Endeplatzhalter P_{FOR1e} der ForEach Aktivität erzeugt werden. Andernfalls wird die Schleife durchlaufen. In den Regeln (5) und (6) kopiert die WoG Assign Aktivität A_1 den Wert der Variablen $V_{WoG\_counter}$ in die Variable $V_{counter}$. Zusätzlich wird in Regel (6) der Beginnplatzhalter für den Schleifenkörper SCO1 erzeugt. Nachdem der Schleifenkörper beendet worden ist, wird in Regel (7) die WoG Assign Aktivität A_2 aktiviert. Sie erhöht den Wert der Zählvariablen $V_{WoG\_counter}$ um eins. Anschließend wird in Regel (8) der neue Schleifendurchlauf gestartet.

(1) $P_{FOR1b} \rightarrow S_{1l}\ V_{WoG\_counter}\ V_{counter}\ P_{E1}\ S_{1r}$

(2) $V_{WoG\_counter}\ P_{E1} \rightarrow V_{WoG\_counter}\ E_1$

(3) $E_1 \rightarrow e_1\ R_{E1}$

(4) $R_{E1} \rightarrow T\ P_{A1}$

(5) $V_{WoG\_counter}\ P_{A1} \rightarrow V_{WoG\_counter}\ A_1$

(6) $V_{counter}\ A_1 \rightarrow V_{counter}\ a_1\ P_{SCO1b}$

(7) $V_{WoG\_counter}\ P_{SCO1e} \rightarrow V_{WoG\_counter}\ A_2$

(8) $V_{WoG\_counter}\ A_2 \rightarrow V_{WoG\_counter}\ a_2\ P_{E1}$

(9) $R_{E1} \rightarrow F$

(10) $T \rightarrow \varepsilon$

(11) $F \rightarrow \varepsilon$

(12) $S_{1l}\ V_{WoG\_counter}\ V_{counter}\ S_{1r} \rightarrow P_{FOR1e}$

Regelmenge 39: WoG Regelmenge für BPEL ForEach FOR1, seriell

```
[01] <nonTerminal xsi:type="tEvaluation"
[02]              id="E1" >
[03]   <condition>($V_WoG_counter < 16) and
[04]              (($V_WoG_counter – 13) < 2)</condition>
[05] </nonTerminal>
```

Listing 62: WoG Evaluation E_1

5.5.7.2. Paralleler Schleifendurchlauf

In Regelmenge 40 wird gezeigt, wie die in Listing 61 vorgestellte BPEL ForEach Aktivität in WoG modelliert wird, wenn das Attribut „parallel“ auf „yes“ gesetzt wird, die Schleife also parallel durchlaufen werden soll. In Regel (1) werden drei Platzhalter P_{S1}, P_{S2} und P_{S3} erzeugt, sie repräsentieren die einzelnen Schleifendurchläufe. Der ebenfalls erzeugte Platzhalter P_1 wird benötigt, um später die BPEL ForEach Aktivität FOR1 geordnet beenden zu können. In Regel (2) wird ein Scope S_1 mit einer Variablen $V_{counter1}$ und dem Beginnplatzhalter für den BPEL Schleifenkörper SCO1 erzeugt. Die Variable $V_{counter1}$ wird mit dem Startwert des Schleifenzählers initialisiert. Regel (3) und (4) beschreiben, dass nach der Ausführung des Schleifenkörpers der Scope S_1 mit der Variablen $V_{counter1}$ gelöscht wird und der Platzhalter P_{1m} erzeugt wird. Die Regeln (5) bis (8) in Kombination mit Regel (3) beschreiben das gleiche Verhalten für die Scopes S_2 und S_3. Der einzige Unterschied besteht in der Initialisierung der Variablen, ihr Wert ist jeweils um eins erhöht. Das bedeutet, $V_{counter2}$ hat den Wert (Startwert + 1) und $V_{counter3}$ hat den Wert (Startwert + 2) bzw. den Endwert des Schleifenzählers. In Regel (9) wird beschrieben, dass die BPEL ForEach Aktivität bereits dann beendet werden kann, wenn zwei Schleifendurchläufe beendet worden sind. Dies wird in BPEL mit dem Element „completionCondition“ beschrieben. Der Platzhalter P_1 definiert den Ort innerhalb des Wortes, an dem die Prozessausführung nach Beendigung der BPEL ForEach Aktivität fortgesetzt wird. Obwohl die BPEL ForEach Aktivität in Regel (9) beendet worden ist, kann einer der drei ursprünglich gestarteten Schleifendurchläufe noch aktiv sein. Der Platzhalter P_2 dient dazu, den von diesem Durchlauf erzeugten Platzhalter P_{1m} zu löschen.

(1) P_{FOR1b} → $P_{S1}\ P_{S2}\ P_{S3}\ P_1$

(2) P_{S1} → $S_{1l}\ V_{counter1}\ P_{SCO1b}\ S_{1r}$

(3) P_{SCO1e} → ε

(4) $S_{1l}\ V_{counter1}\ S_{1r}$ → P_{1m}

(5) P_{S2} → $S_{2l}\ V_{counter2}\ P_{SCO1b}\ S_{2r}$

(6) $S_{2l}\ V_{counter2}\ S_{2r}$ → P_{1m}

(7) P_{S3} → $S_{3l}\ V_{counter3}\ P_{SCO1b}\ S_{3r}$

(8) $S_{3l}\ V_{counter3}\ S_{3r}$ → P_{1m}

(9) $P_{1m}\ P_{1m}\ P_1$ → $P_2\ P_{FOR1e}$

(10) $P_{1m}\ P_2$ → ε

Regelmenge 40: WoG Regelmenge für BPEL ForEach FOR1, parallel

Die beschriebene Umsetzung einer parallelen BPEL ForEach Aktivität in WoG besagt, dass die BPEL ForEach Aktivität gemäß der Abbruchbedingung (BPEL „completionCondition“) vorzeitig beendet wird. Trotzdem laufen in dem gezeigten Beispiel (Regelmenge 40) alle Schleifendurchläufe bis zum Ende durch. In BPEL wird dagegen gefordert, dass, sobald die Abbruchbedingung erfüllt ist, alle noch laufenden Schleifendurchläufe vorzeitig beendet werden. Dies kann in WoG ebenfalls modelliert werden, wird hier aber nicht näher betrachtet.

5.5.7.3. Zusammenfassung

Es gibt zwei grundsätzlich verschiedene Möglichkeiten, eine BPEL ForEach Aktivität in WoG zu übersetzen. Soll sie sequentiell ausgeführt werden, dann wird dies in WoG ähnlich wie eine While Schleife modelliert. Soll die Schleife dagegen parallel ausgeführt werden, dann entspricht die Umsetzung in WoG einer Verzweigung, parallelen Ausführung und anschließender Zusammenführung.

5.5.8. Scope

Eine BPEL Scope Aktivität definiert einen Bereich innerhalb eines Prozesses. Dieser Bereich ist zum einen Lebensbereich für Variablen, PartnerLinks, messageExchanges und CorrelationSets. Zum anderen kann für einen Bereich definiert werden, wie Fehlerbehandlung, Compensation oder Event Handling umgesetzt wird. Eine BPEL Scope Aktivität wird in WoG durch einen WoG Scope dargestellt. Variablen werden auf entsprechende Nichtterminale abgebildet. Die Handler eines BPEL Scopes werden in WoG auf explizite Kontrollflussregeln abgebildet.

```
[01] <scope isolated="yes|no"? exitOnStandardFault="yes|no"?
[02]    standard-attributes>
[03]    standard-elements
[04]    <variables>?
[05]       …
[06]    </variables>
[07]    <partnerLinks>?
[08]       …
[09]    </partnerLinks>
[10]    <messageExchanges>?
[11]       …
[12]    </messageExchanges>
[13]    <correlationSets>?
[14]       …
[15]    </correlationSets>
[16]    <eventHandlers>?
[17]       …
[18]    </eventHandlers>
[19]    <faultHandlers>?
[20]       …
[21]    </faultHandlers>
[22]    <compensationHandler>?
[23]       …
[24]    </compensationHandler>
[25]    <terminationHandler>?
[26]       …
[27]    </terminationHandler>
[28]    activity
[29] </scope>
```

Listing 63: BPEL Scope, XML Struktur

isolated

Listing 63 zeigt den allgemeinen Aufbau einer BPEL Scope Aktivität. Mit dem Attribut „isolated“ kann beeinflusst werden, wie konkurrierende Datenzugriffe parallel ablaufender Scopes behandelt werden. Dieses Attribut wird hier nicht näher betrachtet.

exitOnStandardFault

Mit dem Attribut „exitOnStandardFault“ kann angegeben werden, ob BPEL Standardfehler der normalen Fehlerbehandlung unterworfen werden oder ob sie den Prozess vorzeitig beenden. Ist das Attribut auf „yes“ gesetzt, dann entspricht das Auftreten eines BPEL Standardfehlers der Aktivierung einer BPEL Exit Aktivität. Die Umsetzung einer BPEL Exit Aktivität in WoG wurde bereits in Kapitel 5.4.10 beschrieben. Ist das Attribut auf „no“ gesetzt, dann wird Fehlerbehandlung durchgeführt. Dies wird in Kapitel 5.5.8.1 betrachtet.

Variables

Innerhalb des Elementes „variables“ werden Variablen definiert, die in diesem Scope leben. Für jede BPEL Variable wird in WoG ein entsprechendes Nichtterminal vom Typ Variable erzeugt.

partnerLinks

Innerhalb des Elementes „partnerLinks“ werden PartnerLinks definiert, die innerhalb dieses Scopes verwendet werden können. Dieses Element wird unverändert in das WoG Scope Nichtterminal (linke Scopegrenze) übernommen.

messageExchanges

Innerhalb des Elementes „messageExchanges“ werden MessageExchanges definiert, die innerhalb dieses Scopes verwendet werden können. Dieses Element wird unverändert in das WoG Scope Nichtterminal (linke Scopegrenze) übernommen.

correlationSets

Innerhalb des Elementes „correlationSets“ werden CorrelationSets definiert, die innerhalb dieses Scopes verwendet werden können. Dieses Element wird unverändert in das WoG Scope Nichtterminal (linke Scopegrenze) übernommen.

Handler

Die unterschiedlichen Handler eines BPEL Scopes werden in den folgenden Unterkapiteln beschrieben.

5.5.8.1. Fehlerbehandlung

Tritt innerhalb eines BPEL Scopes ein Fehler auf, so werden zunächst alle laufenden Aktivitäten innerhalb dieses Scopes beendet. Anschließend wird der Termination Handler dieses Scopes aktiviert. Nach dem Ende des Termination Handlers wird der entsprechende Fault Handler aktiviert. Mit der Beendigung des Fault Handlers wird auch der gesamte Scope beendet. Die Modellierung von Fehlern in WoG wurde in Kapitel 3.2.9.3 betrachtet.

In Listing 64 ist der BPEL Scope SCO1 mit zwei Fault Handler und einem Termination Handler dargestellt. Das Element „catch“ in den Zeilen 3 bis 5 beschreibt den Fault Handler für Fehler vom Typ „bpel:invalidExpressionValue“. Er beinhaltet die BPEL Sequence Aktivität SEQ1. Das Element „catchAll“ in den Zeilen 6 bis 8 beschreibt den allgemeinen Fault Handler für alle anderen Fehler, die nicht vom Typ „bpel:invalidExpressionValue“ sind. Dieser beinhaltet die BPEL Flow Aktivität FLO1. In den Zeilen 10 bis 12 ist der Termination Handler definiert. Er beinhaltet die BPEL Empty Aktivität EMP1.

```
[01] <scope name="SCO1">
[02]   <faultHandlers>
[03]     <catch faultName="bpel:invalidExpressionValue">
[04]       <sequence name="SEQ1">…</sequence>
[05]     </catch>
[06]     <catchAll>
[07]       <flow name="FLO1">…</flow>
[08]     </catchAll>
[09]   </faultHandlers>
[10]   <terminationHandler>
[11]     <empty name="EMP1"/>
[12]   </terminationHandler>
[13]       …
[14] </scope>
```

Listing 64: BPEL Scope SCO1

Die Umsetzung dieser Handler in WoG wird in Regelmenge 41 dargestellt. In Regel (1) wird gezeigt, dass für den BPEL Scope SCO1 ein entsprechender WoG Scope erzeugt wird. Dieser Scope S_1 enthält einen Lebensmarker L_{S1} sowie die Killmarkergrenzen K_{S1l} und K_{S1r}. In Kapitel 5.4.10 wurde das vorzeitige Beenden eines Prozesses beschrieben. Ein ähnlicher Mechanismus wird hier verwendet, um einen laufenden Scope vorzeitig zu beenden.

(1) $P_{SCO1b} \rightarrow S_{1l}\ K_{S1l}\ L_{S1}\ V_1\ \ldots\ K_{S1r}\ S_{1r}$

(2) $L_{S1}\ \ldots \rightarrow K_{S1}\ \ldots\ F_1$

(3) $K_{S1l}\ K_{S1}\ K_{S1r}\ F_1 \rightarrow P_{EMP1b}\ F_1$

(4) $P_{EMP1e}\ F_1 \rightarrow P_{SEQ1b}$

(5) $P_{SEQ1e} \rightarrow \varepsilon$

(6) $L_{S1}\ \ldots \rightarrow K_{S1}\ \ldots\ F_2$

(7) $K_{S1l}\ K_{S1}\ K_{S1r}\ F_2 \rightarrow P_{EMP1b}\ F_2$

(8) $P_{EMP1e}\ F_2 \rightarrow P_{FLO1b}$

(9) $P_{FLO1e} \rightarrow \varepsilon$

(10) $S_{1l}\ K_{S1l}\ L_{S1}\ V_1\ K_{S1r}\ S_{1r} \rightarrow P_{SCO1e}$

(11) $S_{1l}\ V_1\ S_{1r} \rightarrow P_{SCO1e}$

Regelmenge 41: WoG Regelmenge für Fehlerbehandlung

In Regel (2) der Regelmenge 41 wird das Auftreten eines Fehlers vom Typ „bpel:invalidExpressionValue“ modelliert, es wird ein entsprechendes Fehlernichtterminal F_1 erzeugt. Dieses wird in Listing 65 dargestellt. Gleichzeitig wird der Lebensmarker L_{S1} gelöscht sowie der Killmarker K_{S1} erzeugt. Damit wird die normale Ausführung des Scopes gestoppt. Innerhalb des Scopes beendet der Killmarker K_{S1} alle laufenden Aktivitäten und löscht alle Nichtterminale, ausgenommen die Variablen dieses Scopes. Nach dem dies durchgeführt worden ist, wird in Regel (3) der Beginnplatzhalter P_{EMP1b} des Termination Handlers erzeugt. Ist dieser beendet, wird in Regel (4) der Beginnplatzhalter P_{SEQ1b} des Fault Handlers für Fehler vom Typ „bpel:invalidExpressionValue“ erzeugt. Ist der Fault Handler beendet (Regel (5)), so kann der Scope S_1 mit Regel (11) beendet werden. Regel (10) modelliert die Beendigung des Scopes für den Fall, dass er normal ausgeführt worden ist und kein Fehler aufgetreten ist. Die Behandlung eines anderen Fehlers, der nicht vom Typ „bpel:invalidExpressionValue“ ist, wird in den Regeln (6)

bis (9) dargestellt. Das Vorgehen ist ähnlich wie in den Regeln (2) bis (5), der Fehler wird dabei durch das Fehlernichtterminal F_2 repräsentiert. Es wird der gleiche Termination Handler aber anschließend ein anderer Fault Handler aktiviert.

Damit wurde, anhand eines Beispiels, gezeigt, wie die Fehlerbehandlung innerhalb eines BPEL Scopes prinzipiell auf WoG abgebildet werden kann. Die vollständige Umsetzung der Fehlerbehandlung eines BPEL Scopes in WoG ist deutlich komplexer und wird in dieser Arbeit nicht behandelt. Es muss für jede Aktivität innerhalb des Scopes analysiert werden, welche Fehler sie erzeugen kann. Dies muss anschließend in WoG modelliert werden. Die Zuordnung von Fehlertypen und Fault Handlern ist in BPEL durch ein nicht triviales Regelwerk beschrieben. Dies muss bei der Umsetzung in WoG ebenfalls beachtet werden.

```
[01] <nonTerminal xsi:type=“tFault"
[02]              id="F1"
[03]              faultName="bpel:invalidExpressionValue"
[04]              borderType="left|right">
[05] </nonTerminal>
```

Listing 65: WoG Fehlernichtterminal F_1

5.5.8.2. Compensation Handling

Nachdem ein BPEL Scope erfolgreich beendet worden ist, wird sein Compensation Handler erzeugt. Das bedeutet, dass dieser ab sofort aufgerufen werden kann. Listing 66 zeigt ein BPEL Scope SCO1 mit der Kindaktivität SEQ1 und einem Compensation Handler mit der Kindaktivität SEQ2. Anhand dieses Beispiels wird im Folgenden skizziert, wie Compensation Handling in WoG modelliert werden kann.

```
[01] <scope name="SCO1">
[02]   <compensationHandler>
[03]     <sequence name="SEQ2">…</sequence>
[04]   </compensationHandler>
[05]   <sequence name="SEQ1">…</sequence>
[06] </scope>
```

Listing 66: BPEL Scope SCO1, Compensation Handler

(1) $P_{SCO1b} \rightarrow S_{1l}\ V_1\ P_{SEQ1b}\ S_{1r}$

(2) $S_{1l}\ V_1\ P_{SEQ1b}\ S_{1r} \rightarrow S_{1l}\ V_1\ P_{CH}\ S_{1r}\ P_{SCO1e}$

(3) $\ldots \rightarrow P_{CHm}$

(4) $P_{CHm}\ P_{CH} \rightarrow P_{SEQ2b}$

(5) $P_{SEQ2e} \rightarrow \varepsilon$

(6) $S_{1l}\ V_1\ S_{1r} \rightarrow \varepsilon$

Regelmenge 42: WoG Regelmenge für Compensation Handling

Regelmenge 42 modelliert das vorgestellte Beispiel in WoG. In Regel (1) wird der Scope SCO1 aktiviert und der Beginnplatzhalter P_{SEQ1b} für seine Kindaktivität SEQ1 erzeugt. In Regel (2) wird die Beendigung des Scopes dargestellt. Besitzt ein Scope einen Compensation Handler, dann wird er nach seiner Abarbeitung nicht gelöscht. Innerhalb des Scopes wird ein Platzhalter P_{CH} erzeugt.

Dieser symbolisiert, dass der Scope SCO1 beendet wurde und der Compensation Handler ab sofort aufgerufen werden kann. Die Aktivierung des Compensation Handlers wird in Regel (3) und (4) dargestellt. In Regel (3) wird ein mobiler Platzhalter P_{CHm} erzeugt. Regel (4) beschreibt, dass es mit diesem mobilen Platzhalter möglich ist, den Compensation Handler des Scopes SCO1 zu aktivieren. Regel (5) modelliert das Beenden des Compensation Handlers. In Regel (6) wird beschrieben, dass der Scope SCO1 erst dann gelöscht werden darf, nachdem sein Compensation Handler abgearbeitet wurde.

In Regelmenge 42 wurde skizziert, wie Compensation Handling in WoG modelliert werden kann. Es wurde ein einfaches Beispiel betrachtet. Für die Modellierung von Compensation Handling in komplexeren Prozessen müssen noch weitere Aspekte beachtet werden. In dieser Arbeit wird Compensation Handling nicht näher betrachtet.

5.5.8.3. Event Handling

BPEL Event Handler beschreiben Aktivitäten, die parallel zur Ausführung eines BPEL Scopes aktiviert werden können. Ähnlich wie bei einer BPEL Pick Aktivität können Event Handler durch eingehende Nachrichten oder zeitliche Bedingungen aktiviert werden. Ein BPEL Event Handler wird in WoG auf eine WoG In oder eine WoG Wait Aktivität abgebildet.

```
[01] <eventHandlers>?
[02]   <onEvent partnerLink="NCName"
[03]              portType="QName"?
[04]              operation="NCName"
[05]              ( messageType="QName" | element="QName" )?
[06]              variable="BPELVariableName"?
[07]              messageExchange="NCName"?>*
[08]     <correlations>?  …  </correlations>
[09]     <fromParts>?  …  </fromParts>
[10]     <scope …>…</scope>
[11]   </onEvent>
[12]   <onAlarm>*
[13]     (
[14]       <for expressionLanguage="anyURI"?>duration-expr</for>
[15]       |
[16]       <until expressionLanguage="anyURI"?>deadline-expr</until>
[17]     )?
[18]     <repeatEvery expressionLanguage="anyURI"?>?
[19]       duration-expr
[20]     </repeatEvery >
[21]     <scope …>…</scope>
[22]   </onAlarm>
[23] </eventHandlers>
```

Listing 67: BPEL Event Handler, XML Struktur

Listing 67 zeigt den Aufbau der Event Handler eines Scopes. Die Lebensdauer eines Event Handlers ist gebunden an die Lebensdauer des Scopes, in dem er definiert ist. Jedes „onEvent“ Element beschreibt einen Event Handler, der durch eingehende Nachrichten aktiviert wird. Diese Event Handler können beliebig oft aktiviert werden. Jedes „onAlarm“ Element beschreibt einen Event Handler, der durch eine zeitliche Bedingung aktiviert wird. Von dieser Bedingung hängt ab, wie oft der Event Handler aktiviert wird. Im Folgenden wird für beide Arten von Event Handler beschrieben, wie sie auf WoG abgebildet werden können.

onEvent, onAlarm

Listing 68 zeigt einen BPEL Scope SCO1 mit der Kindaktivität FLO1 und zwei Event Handlern. In den Zeilen 3 bis 8 ist der erste Event Handler dargestellt, er wird durch eingehende Nachrichten aktiviert. Die Attribute „partnerLink“ und „operation“ beschreiben die Schnittstelle, über die die Nachrichten empfangen werden. Wird eine Nachricht empfangen, dann wird eine Variable erzeugt und die Nachricht darin abgelegt. Name und Typ dieser Variable sind in Zeile 5 und 6 hinterlegt. Anschließend wird der Scope SCO2 aktiviert (Zeile 7). Dieser Event Handler bleibt auch nach dem Empfangen einer Nachricht aktiv und kann weitere eingehende Nachrichten, wie beschrieben, verarbeiten. Dies bedeutet insbesondere, dass es mehrere Instanzen der Aktivität SCO2 geben kann, die parallel aktiv sind.

In den Zeilen 9 bis 13 ist der zweite Event Handler dargestellt, er wird durch zeitliche Bedingungen aktiviert. Das Element „until“ beschreibt den Zeitpunkt, an dem der Event Handler das erste Mal aktiviert wird. Das Element „repeatEvery“ gibt an, dass er anschließend regelmäßig wiederholt aktiviert wird. Das Element definiert die Dauer zwischen den einzelnen Aktivierungen. Bei jeder Aktivierung dieses Event Handlers wird seine in Zeile 12 angegebene Kindaktivität SCO3 ausgeführt.

```
[01] <scope name="SCO1">
[02]   <eventHandlers>
[03]     <onEvent partnerLink="PL1"
[04]              operation="checkOrder"
[05]              messageType="orderInfoMsg"
[06]              variable="orderInfo">
[07]       <scope name="SCO2">…</scope>
[08]     </onEvent>
[09]     <onAlarm>
[10]       <until>’2011-08-31T12:00:00’</until>
[11]       <repeatEvery>’PT1H’</repeatEvery>
[12]       <scope name="SCO3">…</scope>
[13]     </onAlarm>
[14]   </eventHandlers>
[15]   <flow name="FLO1" …>…</flow>
[16] </scope>
```

Listing 68: BPEL Event Handler, Beispiel

Die Modellierung des BPEL Scopes SCO1 in WoG wird in Regelmenge 43 dargestellt. In Regel (1) aktiviert der Beginnplatzhalter P_{SCO1b} den WoG Scope S_1 und mehrere Platzhalter. Die Platzhalter P_{I1_start1} und P_{W1_start} modellieren mit den Regeln (2) bis (4) die Aktivierung der beiden Event Handler. Nachdem dies geschehen ist, kann mit dem Platzhalter P_1 in Regel (5) die Kindaktivität FLO1 des Scopes SCO1 aktiviert werden.

In Regel (1) werden bei der Erzeugung des WoG Scopes S_1 zusätzlich der Lebensmarker L_{S1} sowie die Killmarkergrenzen K_{S1l} und K_{S1r} für diesen Scope erzeugt. Diese werden für die Beendigung des Scopes benötigt. Nachdem die Kindaktivität FLO1 des Scopes S_1 beendet wurde, wird in Regel (15) der Lebensmarker L_{S1} gelöscht und der Killmarker K_{S1} erzeugt. Er wird benötigt, da die Event Handler des Scopes parallel zur Kindaktivität aktiv sind. Mit dem Beenden der Kindaktivität müssen auch die Event Handler beendet werden. Dies geschieht durch den Killmarker. Anschließend kann mit Regel (16) der Killmarker gelöscht und mit Regel (17) der gesamte Scope S_1 beendet und der Endeplatzhalter P_{SCO1e} erzeugt werden.

Die Aktivierung des „onEvent“ Event Handlers wird in Regel (2) und (3) modelliert. In Regel (2) wird zunächst ein WoG Scope S_2 mit der Variablen $V_{orderInfo}$ und den Platzhaltern P_{I1_start2} und P_2 erzeugt. In der Variablen $V_{orderInfo}$ wird die empfangene Nachricht abgelegt. Der Platzhalter P_2

markiert den Ort, an dem nachfolgende Instanzen dieses Event Handlers erzeugt werden können. Der Platzhalter P_{I1_start2} aktiviert in Regel (3) die WoG In Aktivität I_1 sowie den Platzhalter P_{1m}. Die Aktivität I_1 wartet auf eingehende Nachrichten. Der Platzhalter P_{1m} signalisiert, dass damit ein Event Handler aktiv ist. Ist auch der „onAlarm“ Event Handler in Regel (4) aktiviert worden, kann die Kindaktivität FLO1 des Scopes SCO1 mit Regel (5) aktiviert werden.

Empfängt die WoG In Aktivität I_1 eine Nachricht, so wird sie in Regel (6) beendet und die Nachricht in der Variablen $V_{orderInfo}$ abgespeichert. Gleichzeitig werden die Platzhalter P_{SCO2b} und P_{2m} erzeugt. Der Beginnplatzhalter P_{SCO2b} aktiviert die Kindaktivität SCO2 des „onEvent“ Event Handlers. Der Platzhalter P_{2m} wird in Regel (7) verwendet, um eine weitere Instanz des „onEvent“ Event Handlers zu erzeugen. Dies geschieht ähnlich wie in Regel (2). In Regel (8) wird eine weitere WoG In Aktivität I_1 aktiviert. Die Regel (9) und (10) beschreiben das Beenden einer Instanz des „onEvent“ Event Handlers.

(1) $P_{SCO1b} \rightarrow S_{1I}\ K_{S1I}\ L_{S1}\ P_{I1\_start1}\ P_{W1\_start}\ P_1\ K_{S1r}\ S_{1r}$

(2) $L_{S1}\ P_{I1\_start1} \rightarrow L_{S1}\ S_{2I}\ V_{orderInfo}\ P_{I1\_start2}\ S_{2r}\ P_2$

(3) $L_{S1}\ P_{I1\_start2} \rightarrow L_{S1}\ I_1\ P_{1m}$

(4) $L_{S1}\ P_{W1\_start} \rightarrow L_{S1}\ W_1\ P_{1m}$

(5) $L_{S1}\ P_{1m}\ P_{1m}\ P_1 \rightarrow L_{S1}\ P_{FLO1b}$

(6) $L_{S1}\ V_{orderInfo}\ I_1 \rightarrow L_{S1}\ V_{orderInfo}\ i_1\ P_{SCO2b}\ P_{2m}$

(7) $L_{S1}\ P_{2m}\ P_2 \rightarrow L_{S1}\ S_{2I}\ V_{orderInfo}\ P_{I1}\ S_{2r}\ P_2$

(8) $L_{S1}\ P_{I1} \rightarrow L_{S1}\ I_1$

(9) $L_{S1}\ P_{SCO2e} \rightarrow L_{S1}$

(10) $L_{S1}\ S_{2I}\ V_{orderInfo}\ S_{2r} \rightarrow L_{S1}$

(11) $L_{S1}\ W_1 \rightarrow L_{S1}\ w_1\ P_{SCO3b}\ P_{W2}$

(12) $L_{S1}\ P_{W2} \rightarrow L_{S1}\ W_2$

(13) $L_{S1}\ W_2 \rightarrow L_{S1}\ w_2\ P_{SCO3b}\ P_{W2}$

(14) $L_{S1}\ P_{SCO3e} \rightarrow L_{S1}$

(15) $L_{S1}\ P_{FLO1e} \rightarrow K_{S1}$

(16) $K_{S1I}\ K_{S1}\ K_{S1r} \rightarrow \varepsilon$

(17) $S_{1I}\ S_{1r} \rightarrow P_{SCO1e}$

Regelmenge 43: WoG Regelmenge für Event Handling

Die Aktivierung des „onAlarm“ Event Handlers wird in Regel (4) modelliert. Der Platzhalter P_{W1_start} aktiviert in dieser Regel die WoG Wait Aktivität W_1 sowie den Platzhalter P_{1m}. Die Aktivität W_1 wartet auf den Zeitpunkt, der im BPEL „onAlarm“ Event Handler durch das Element „until“ angegeben ist. Der Platzhalter P_{1m} signalisiert, dass damit ein Event Handler aktiv ist. In Regel (11) wird die Aktivität W_1 beendet und der Beginnplatzhalter P_{SCO3b} sowie der Platzhalter P_{W2} erzeugt. Der Beginnplatzhalter P_{SCO3b} aktiviert die BPEL Kindaktivität SCO3 des „„onAlarm“ Event Handlers. Der Platzhalter P_{W2} aktiviert in Regel (12) die WoG Wait Aktivität W_2. Diese wartet so lange, wie im BPEL „onAlarm“ Event Handler durch das Element „repeatEvery“ angegeben ist. Anschließend wird die Aktivität W_2 in Regel (13) beendet und erneut der Beginnplatzhalter P_{SCO3b} sowie der Platzhalter P_{W2} erzeugt. Regel (14) beschreibt das Beenden einer Instanz des „onAlarm“ Event Handlers.

5.5.8.4. Zusammenfassung

Eine BPEL Scope Aktivität wird abgebildet auf einen WoG Scope. Die in einem BPEL Scope definierten Variablen werden auf entsprechende WoG Variablen abgebildet. Die Elemente „partnerLinks“, „messageExchanges“ und „correlationSets“ werden unverändert vom BPEL Scope in den WoG Scope übernommen. In Listing 69 ist der allgemeine Aufbau eines WoG Scope Nichtterminals abgebildet. Ein WoG Scope wird durch seine linke und rechte Grenze repräsentiert.

```
[01] <nonTerminal xsi:type="tScopeBorder"
[02]              id="NCName"
[03]              name="NCName"?
[04]              borderType="left|right" >
[05]    <partnerLinks>?
[06]       …
[07]    </partnerLinks>
[08]    <messageExchanges>?
[09]       …
[10]    </messageExchanges>
[11]    <correlationSets>?
[12]       …
[13]    </correlationSets>
[14] </nonTerminal>
```

Listing 69: WoG Scope, XML Struktur

Die Handler eines BPEL Scopes werden in WoG auf expliziten Kontrollfluss innerhalb des WoG Scopes abgebildet. Sowohl für den Event Handler als auch für den Fault Handler wurde die Abbildung auf WoG beschrieben und an Beispielen gezeigt. Beim Fault Handler wurde die detaillierte Kontrollflusslogik, d.h. wo im Prozess welche Fehler auftreten können, nicht näher betrachtet. Es wurde aber gezeigt, wie dies in WoG modelliert werden kann. Für den Compensation Handler wurde lediglich skizziert, wie er auf WoG abgebildet werden kann.

5.6. Zusammenfassung

Sowohl BPEL als auch WoG verwenden als Datenmodell XML Schema und kommunizieren über Web Services. Bei der Transformation von BPEL nach WoG werden diesbezügliche Attribute und Elemente in den meisten Fällen unverändert nach WoG übernommen.

Basierend auf der Blockstruktur von BPEL wurde eine Methode entwickelt, mit deren Hilfe die Übersetzung von BPEL nach WoG modular aufgebaut werden kann. Jede BPEL Aktivität wird auf eine Regelmenge abgebildet. So eine Regelmenge besitzt klar definierte Schnittstellen in Form von Beginn- und Endeplatzhaltern. Das Erzeugen eines Beginnplatzhalters aktiviert diese Regelmenge, repräsentiert somit die Aktivierung der entsprechenden BPEL Aktivität. Nachdem die Regelmenge abgearbeitet worden ist, wird ihr Endeplatzhalter erzeugt. Dieser signalisiert die Beendigung der entsprechenden BPEL Aktivität. Bei der Übersetzung von strukturierten BPEL Aktivitäten nach WoG werden die Kindaktivitäten nicht mit übersetzt. Sie werden lediglich über ihre Beginn- und Endeplatzhalter in die erzeugte Regelmenge eingebunden.

Die Übersetzung von BPEL Aktivitäten nach WoG wurde in zwei Teile untergliedert. Zunächst wurden alle BPEL Basisaktivitäten betrachtet, anschließend wurde die Übersetzung strukturierter BPEL Aktivitäten beschrieben.

BPEL Basis Aktivitäten werden in vielen Fällen auf eine WoG Aktivität abgebildet, welche eine ähnliche Funktionalität wie die BPEL Aktivität besitzt. Eine BPEL Invoke Aktivität wird beispielsweise auf eine WoG Call Aktivität abgebildet. Beide Aktivitäten beschreiben den Aufruf eines Web Services. BPEL Basisaktivitäten beinhalten aber oft mehr als nur eine einzelne

Funktionalität. Eine BPEL Invoke Aktivität beschreibt beispielsweise implizit immer auch einen Scope. Die Nutzung der Elemente „fromParts“ oder „toParts“ innerhalb einer Aktivität beschreibt implizit die Erzeugung einer temporären Variablen. Das führt dazu, dass eine BPEL Basis Aktivität nicht nur auf eine WoG Aktivität, sondern in vielen Fällen auf mehrere WoG Aktivitäten mit den entsprechenden Regeln abgebildet wird.

Strukturierte BPEL Aktivitäten beschreiben den Kontrollfluss zwischen ihren Kindaktivitäten. Ihre Übersetzung nach WoG erzeugt im Allgemeinen komplexere und größere Regelmengen als die Übersetzung von Basisaktivitäten. Kontrollfluss wird in WoG immer explizit modelliert, während er in BPEL in vielen Fällen implizit modelliert wird. Ein Beispiel dafür ist die ForEach Aktivität. Abhängig vom Attribut „parallel“ kann sie komplett unterschiedlichen Kontrollfluss beschreiben (Sequenz oder Parallelität). Das konkrete Verhalten wird in BPEL nicht explizit modelliert, sondern durch den BPEL Standard definiert und beschrieben.

Die Übersetzung einer BPEL Aktivität nach WoG wurde zunächst allgemein beschrieben und in vielen Fällen mit Hilfe von Beispielen verdeutlicht. Bei der erstmaligen Verwendung eines WoG Nichtterminals wurde seine XML Struktur vorgestellt.

Die Abbildung der BPEL Flow und BPEL Scope Aktivität wurde nicht vollständig beschrieben. Bei der BPEL Flow Aktivität wurden „Cross Boundary Links“ nicht betrachtet. Die Übersetzung des „Compensation Handlers“ einer BPEL Scope Aktivität wurde skizziert, aber nicht im Detail ausgearbeitet. „Isolated Scopes“ wurden in dieser Arbeit nicht betrachtet.

6 Implementierung der Transformation

6.1. Verwendete Technologien

6.1.1. Eclipse

Eclipse[1] ist eine in Java entwickelte Anwendung, die durch ihre modulare Struktur erweiterbar und anpassbar ist. Die Module, aus denen Eclipse besteht und durch die es erweitert werden kann, bezeichnet man als Plugin. Der Kern von Eclipse, die sogenannte Eclipse Plattform, kann als Basis zur Entwicklung von beliebigen Anwendungen dienen [IBM06].

6.1.2. EMF

Das Eclipse Modeling Framework (EMF)[2] ist ein Java Framework, welches die Entwicklung von Java Anwendungen auf der Basis von strukturierten Modellen unterstützt. Das Ausgangsmodell kann in verschiedenen Formaten vorliegen. Es können Modelle eingelesen werden, die als UML, XML oder annotierte Java Interfaces vorliegen. Das eingelesene Modell wird in ein EMF Modell konvertiert. Dieses Modell kann parametrisiert und angepasst werden. Aus dem EMF Modell wird anschließend Java Code generiert [IBM04].

6.1.3. BPEL Designer

Der Eclipse BPEL Designer[3] ist eine Anwendung zur graphischen Erstellung von BPEL Prozessen. Der BPEL Designer ist als Eclipse Plugin implementiert. Seine Architektur folgt dem Model View Controller (MVC) Prinzip. Das dem BPEL Designer zugrunde liegende Modell bildet den BPEL Standard ab und wurde mit EMF umgesetzt.

6.1.4. JAXB

Java Architecture for XML Binding (JAXB) ist eine Java Schnittstelle zur Deserialisierung von XML Dokumenten in Java Objekte und umgekehrt zur Serialisierung von Java Objekten in XML Dokumente. Diese Schnittstelle wird unter anderem von der Java JRE implementiert. Sowohl die Deserialisierung als auch die Serialisierung kann parametrisiert und angepasst werden. Zusätzlich bildet JAXB XML Schema Definitionen auf Java Klassen ab [Dau07].

[1] http://www.eclipse.org/

[2] http://www.eclipse.org/modeling/emf/

[3] http://www.eclipse.org/bpel/

6.2. Umsetzung

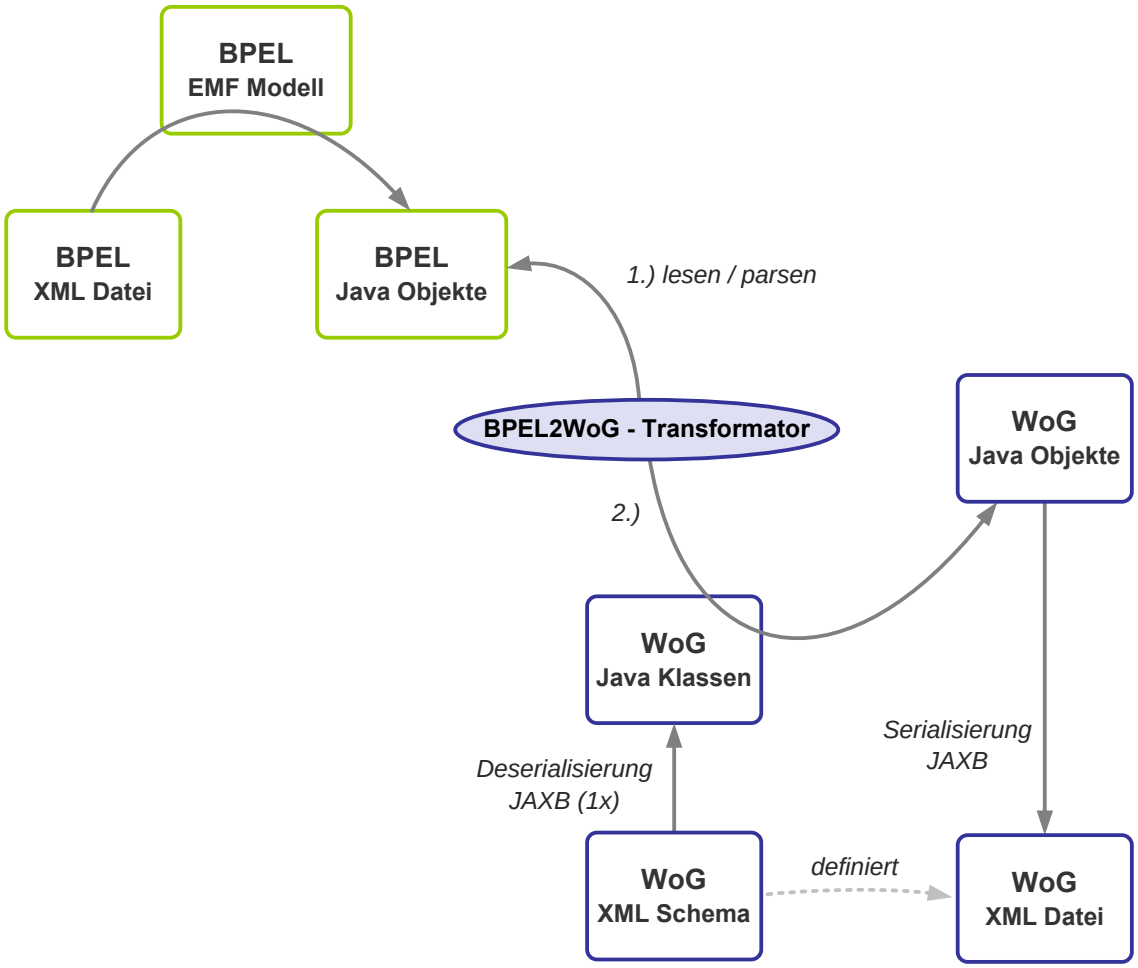


Abbildung 39: Übersicht

Abbildung 39 gibt einen Überblick über die Implementierung des Transformators von BPEL nach WoG. Der BPEL2WoG Transformator verwendet das BPEL EMF Modell des Eclipse BPEL Designers. Dieses Modell bildet BPEL auf Java Klassen ab. Zusätzlich werden weitere Funktionalitäten des Eclipse BPEL Designers verwendet, um BPEL XML Dateien in entsprechende Java Objekte zu überführen.

In Kapitel 4 wurde ein XML Dateiformat für WoG Grammatiken vorgestellt. Dieses Dateiformat wird durch ein XML Schema definiert. Aus diesem XML Schema wird mit JAXB eine Java Klassenstruktur für die Darstellung von WoG Grammatiken erzeugt.

Der BPEL2WoG Transformator bekommt einen BPEL Prozess in Form einer Java Objektstruktur übergeben. Basierend auf den aus dem WoG XML Schema erzeugten Klassen bildet der Transformator den BPEL Prozess auf eine WoG Grammatik ab. Die so entstandene Objektstruktur wird anschließend mit JAXB serialisiert und als WoG XML Datei abgespeichert.

Der Transformator implementiert im Wesentlichen das in Kapitel 5 beschriebene Vorgehen zur Transformation von BPEL nach WoG. Der Transformator durchläuft den BPEL Prozess in einem Breitendurchlauf. Dabei wird jede BPEL Aktivität in eine WoG Regelmenge übersetzt. Für jeden BPEL Aktivitätstyp gibt es eine Methode für die Übersetzung nach WoG.

Listing 70 zeigt die zentrale Methode des BPEL2WoG Transformators. Dieser bekommt einen BPEL Prozess übergeben und erzeugt daraus eine WoG Grammatik (Zeile 1). Im Breitendurchlauf

werden alle noch zu übersetzenden BPEL Aktivitäten in einer „todo“ Liste verwaltet. Diese Liste wird als FiFo (First in, First out) umgesetzt und in Zeile 2 initialisiert. In Zeile 3 wird eine leere WoG Grammatik erzeugt und initialisiert. Anschließend werden in Zeile 4 die globalen Definitionen des BPEL Prozesses nach WoG überführt. Dabei wird die Kindaktivität des BPEL Prozesses als erstes Element in die „todo“ Liste eingefügt. Die Übersetzung der einzelnen Aktivitäten des BPEL Prozesses findet innerhalb der while Schleife statt (Zeile 6 bis 23). In jedem Durchlauf wird genau eine BPEL Aktivität aus der „todo“ Liste entfernt (Zeile 7) und in eine WoG Regelmenge übersetzt. Wie bereits erwähnt, gibt es für jeden BPEL Aktivitätstypen eine Methode zur Übersetzung in eine WoG Regelmenge. Bei der Übersetzung von strukturierten BPEL Aktivitäten werden die Kindaktivitäten der „todo“ Liste hinzugefügt, so dass sie in späteren Schleifendurchläufen ebenfalls übersetzt werden. Wenn die „todo“ Liste leer ist, bedeutet dies, dass alle Aktivitäten des BPEL Prozesses übersetzt worden sind. Die Schleife wird verlassen und das erzeugte WoG Grammatik Objekt wird zurückgegeben.

```
[01] public TWoGGrammar transformToWoG(Process p) {
[02]   todo = new FiFo<Activity>();
[03]   wog = Util.createWoGGrammar();
[04]   procToWoG(p);
[05]
[06]   while (todo.isEmpty() == false) {
[07]     Activity bpelActivity = todo.get();
[08]     if (bpelActivity instanceof Assign) {
[09]       assToWoG((Assign) bpelActivity);
[10]     } else if (bpelActivity instanceof Invoke) {
[11]       invToWoG((Invoke) bpelActivity);
[12]     } else if (bpelActivity instanceof Receive) {
[13]       recToWoG((Receive) bpelActivity);
[14]     } else if (bpelActivity instanceof Reply) {
[15]       repToWoG((Reply) bpelActivity);
[16]     } else if (bpelActivity instanceof Scope) {
[17]       scoToWoG((Scope) bpelActivity);
[18]     } else if (bpelActivity instanceof Sequence) {
[19]       seqToWoG((Sequence) bpelActivity);
[20]     } else if (…) {
[21]       …
[22]     }
[23]   }
[24]   return wog;
[25] }
```

Listing 70: BPEL2WoG Transformator

Abbildung 40 zeigt die Funktionsweise des BPEL2WoG Transformators am Beispiel des in Abbildung 41 dargestellten BPEL Prozesses. In Schritt (1) werden die Aktivierungs- und Beendigungsregel für den gesamten Prozess erzeugt. In Schritt (2) wird die Regelmenge für den Prozessscope erzeugt. Die Kindaktivität des Prozesses wird in die „todo“ Liste eingefügt. In Schritt (3) beginnt die Abarbeitung der „todo“ Liste. Die BPEL Aktivität SEQ1 wird aus der „todo“ Liste entfernt und in eine entsprechende WoG Regelmenge übersetzt. Dabei werden ihre Kindaktivitäten in die „todo“ Liste eingefügt. In Schritt (4) wird die BPEL Aktivität REC1 aus der „todo“ Liste entfernt und in eine WoG Regelmenge übersetzt. Die Aktivität REC1 ist eine Basisaktivität und besitzt somit keine Kindaktivitäten. Dementsprechend werden in diesem Schritt keine neuen Aktivitäten in die „todo“ Liste eingefügt. In Schritt (7) wird die letzte BPEL Aktivität aus der „todo“ Liste entfernt und übersetzt. Da in diesem Schritt keine neue BPEL Aktivität in die Liste eingefügt wird, ist die Übersetzung des gesamten BPEL Prozesses nach WoG nach der Abarbeitung von Schritt (7) beendet.

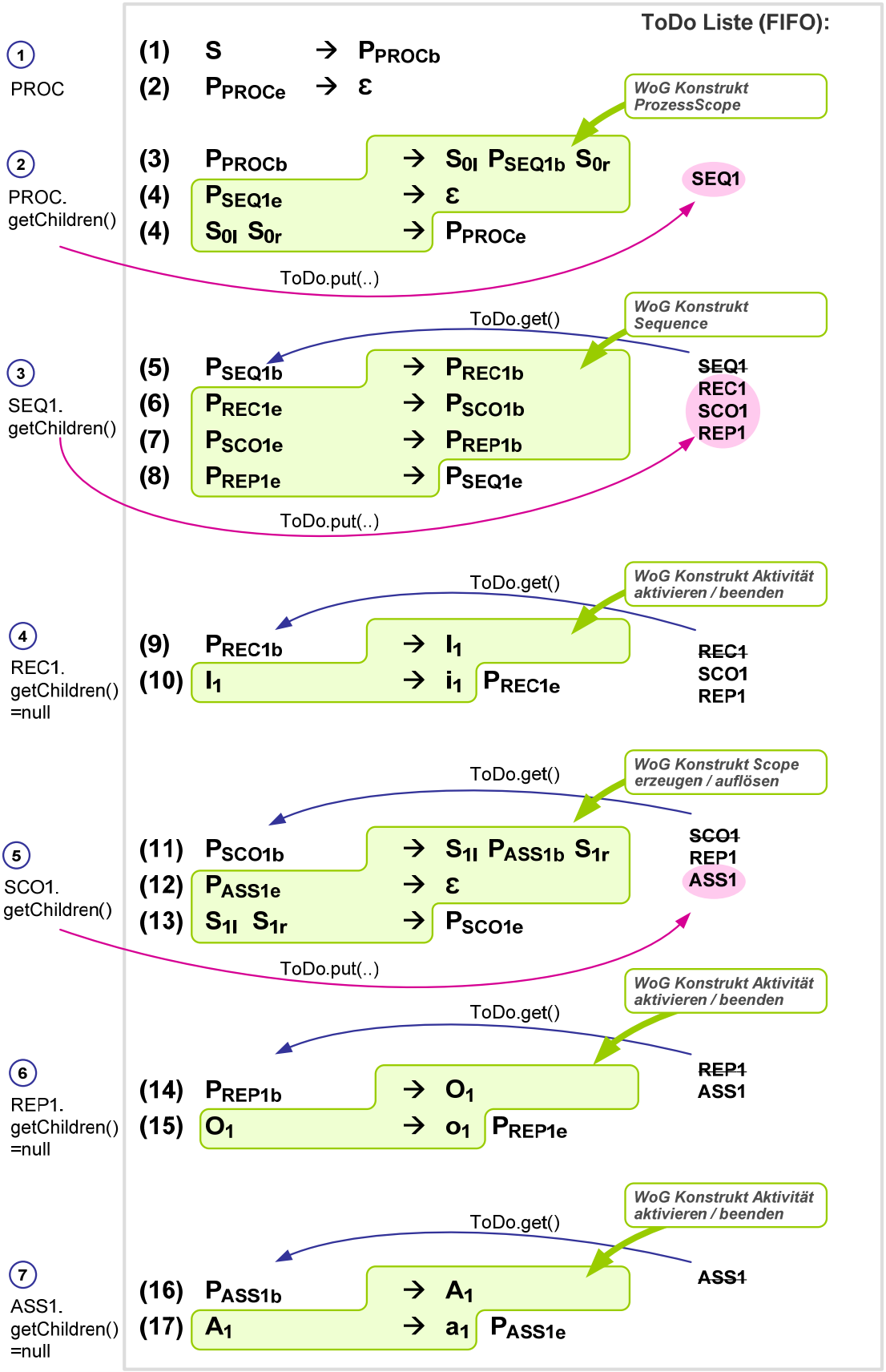


Abbildung 40: Funktionsweise des Transformators, Beispiel

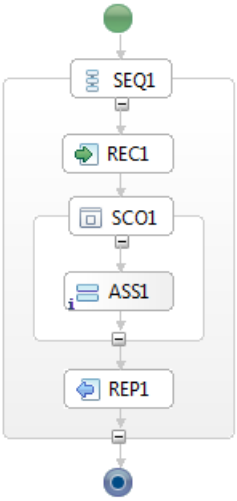


Abbildung 41: BPEL Beispielprozess

Der Transformator ist so entwickelt, dass er zum Einen alleine lauffähig ist und zum Beispiel über Kommandozeile aufgerufen werden kann. Zum Anderen kann er als Eclipse Plugin installiert werden. BPEL Dateien können dann direkt über das Kontextmenü (Rechtsklick) in WoG Grammatiken umgewandelt werden.

7 Zusammenfassung und Ausblick

Die wesentlichen Beiträge dieser Arbeit sind das Metamodell für ausführbare Workflow Grammatiken sowie eine Transformation von BPEL Prozessmodellen in WoG Grammatiken. Des Weiteren wurde ein Dateiformat für WoG Grammatiken definiert sowie die beschriebene Transformation von BPEL nach WoG implementiert.

Die grundlegende Idee beim Entwurf von WoG ist die Entwicklung einer allgemeinen Workflow Modellierungssprache. Es soll erreicht werden, dass möglichst viele unterschiedliche Workflow Modellierungssprachen auf WoG abbildbar sind. Diese Motivation wurde zu Beginn von Kapitel 3 beschrieben. Daneben wurde der Zusammenhang zwischen einer Formalen Grammatik und einem Prozessmodell erläutert. Die Regeln der Grammatik beschreiben die Struktur des Prozessmodells. Aktivitäten werden durch Terminale und Nichtterminale repräsentiert. Eine Ableitung eines Wortes beschreibt einen konkreten Prozessdurchlauf. Das Wort selbst repräsentiert eine beendete Prozessinstanz.

Im ersten Teil der Beschreibung des WoG Metamodells wurden die verschiedenen Aktivitätstypen definiert. Im zweiten Teil wurden die WoG Modellierungskonstrukte vorgestellt. Sowohl Kontroll- als auch Datenfluss werden durch Regeln modelliert. Der Kontrollfluss wird explizit, der Datenfluss implizit beschrieben. Es gibt drei prinzipiell unterschiedliche Arten, Kontrollfluss zu modellieren. Aktivitäten können sequentiell, parallel oder abhängig von Bedingungen ausgeführt werden. Bedingter Kontrollfluss kann in WoG abhängig von Booleschen Bedingungen, Fehlern oder dem Empfang von Nachrichten modelliert werden. In den verschiedenen Modellierungskonstrukten wurden zusätzlich zu den Aktivitäten noch weitere Nichtterminaltypen eingeführt. Alle WoG Nichtterminaltypen wurden am Ende von Kapitel 3 noch einmal gesammelt dargestellt und kurz beschrieben.

Die WoG Aktivitätstypen wurden in Kapitel 3 zunächst allgemein beschrieben. Die Aktivität Assign wurde beispielsweise als Zuweisung eingeführt. Es wurde aber noch nicht spezifiziert, wie genau eine Zuweisung in WoG definiert ist. Da WoG wesentliche Konzepte für Kommunikation und Datenmodell von BPEL übernimmt, wurde die genaue Ausgestaltung der einzelnen Aktivitätstypen während der Transformation von BPEL nach WoG in Kapitel 5 beschrieben.

Für die Darstellung von WoG Grammatiken wurde ein XML basiertes Dateiformat definiert. In Kapitel 4 wurde der Aufbau von WoG XML Dateien beschrieben. Die Darstellung der einzelnen Nichtterminaltypen wurde anschließend in Kapitel 5 gezeigt.

BPEL ist eine mächtige und ausdrucksstarke Workflowsprache. Durch die Transformation von BPEL nach WoG wurde gezeigt, dass das entwickelte WoG Metamodell für die Modellierung komplexer Prozessstrukturen geeignet ist.

Für die Transformation von BPEL nach WoG wurde ein Verfahren entwickelt, mit dem die Aktivitäten eines BPEL Prozessmodells einzeln auf WoG Regelmengen abgebildet werden können. Dabei wird jede Aktivität losgelöst vom umgebenden Prozessmodell betrachtet. Die Transformation eines BPEL Prozessmodells reduziert sich damit auf die Transformation der einzelnen BPEL Aktivitäten dieses Prozessmodells. Im weiteren Verlauf von Kapitel 5 wurde für jeden BPEL Aktivitätstypen eine Transformation nach WoG beschrieben. Die vorgestellte Transformation von BPEL nach WoG ist nicht vollständig. „Isolated Scopes“ und „Cross Boundary Links“ wurden nicht betrachtet. Die Transformation von BPEL Compensation Handler nach WoG wurde skizziert, aber nicht im Detail ausgearbeitet.

Zum Schluss der Arbeit wurde in Kapitel 6 die Implementierung des BPEL2WoG Transformators beschrieben. Dieser liest eine BPEL Datei ein, transformiert das BPEL Prozessmodell in eine

Workflow Grammatik und speichert diese als WoG XML Datei ab. Die Implementierung basiert auf dem in Kapitel 5 vorgestellten Entwurf der Transformation von BPEL nach WoG.

Im Folgenden soll ein kurzer Ausblick gegeben werden, welche Aspekte der hier vorgestellten Arbeit in Zukunft weiter verfolgt werden können. Die Transformation von BPEL nach WoG kann vervollständigt werden. Des Weiteren können auch für andere Workflowsprachen Transformationen nach WoG definiert werden. Durch die Betrachtung mehrerer, konzeptionell unterschiedlicher Sprachen kann der Entwurf des WoG Metamodell verifiziert oder andernfalls korrigiert oder erweitert werden.

In [AH10] wird ein Überblick über allgemeine Modellierungskonstrukte zur Darstellung von Prozessmodellen gegeben. Dies erfolgt über die Definition von Mustern, den sogenannten „Workflow Patterns“[4]. Diese Muster beschreiben abstrakt, welche Modellierungskonstrukte es in Workflowsprachen gibt. Eine allgemeine Möglichkeit zu zeigen, dass WoG Prozessmodelle von beliebigen Workflowsprachen modellieren kann, ist ein Vergleich des WoG Metamodells mit Workflow Patterns.

[4] http://www.workflowpatterns.com/

8 Referenzen

[Aal98] van der Aalst, W. M. P. (1998): The application of Petri nets to workflow management. In: J. Circ. Syst. and Comp 8 (1), S. 21–66.

[AH10] van der Aalst, Wil; ter Hofstede, Arthur: Workflow Patterns. Eindhoven University of Technology, Queensland University of Technology.
Online verfügbar unter http://www.workflowpatterns.com/

[BPEL07] Organization for the Advancement of Structured Information Standards (OASIS): Web Services Business Process Execution Language Version 2.0 (WSBPEL). 2007.
Online verfügbar unter
http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[Cla03] Claus, Volker (2003): Duden, Informatik. Ein Fachlexikon für Studium und Praxis. Mannheim; Leipzig; Wien; Zürich: Dudenverlag.

[Dau07] Daum, Berthold (Hg.) (2007): Java 6. Programmieren mit der Java Standard Edition. München; Boston [u.a.]: Addison-Wesley.

[Die05] Dieterich, Ernst-Wolfgang (2005): Assembler. Grundlagen der PC-Programmierung. überarbeitete Auflage: Oldenbourg Wissenschaftsverlag.

[DRV00] Darte, Alain; Robert, Yves; Vivien, Frédéric (2000): Scheduling and automatic parallelization. Boston: Birkhauser.

[FZ09] Finger, Patrick; Zeppenfeld, Klaus (2009): SOA und WebServices. Berlin ; Heidelberg: Springer.

[Hau11] Haupt, Florian (2011): Ausführung von Grammatikbasierten Prozessmodellen in einer Cloud Umgebung. Diplomarbeit. Universität Stuttgart, Stuttgart. Institut für Architektur von Anwendungssystemen.

[IBM04] IBM Corporation (2004): Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework. An IBM Redbooks publication.
Online verfügbar unter
http://www.redbooks.ibm.com/abstracts/sg246302.html?Open

[IBM06] IBM Corporation: Eclipse Platform. Technical Overview. 2006.
Online verfügbar unter
http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf

[JMS06] Juric, Matjaz; Mathew, Benny; Sarang, Poornachandra (2006): Business process execution language for Web services. 2nd. Birmingham: Packt Publishing.

[KS09] Küveler, Gerd; Schwoch, Dietrich (2009): Informatik für Ingenieure und Naturwissenschaftler. 6. Aufl. Wiesbaden: Vieweg + Teubner in GWV Fachverlage GmbH.

[LR00] Leymann, Frank; Roller, Dieter (2000): Production workflow. Concepts and techniques. Upper Saddle River, N.J: Prentice Hall PTR.

[OVA+07] Ouyang, C.; Verbeek, E.; van der Aalst, W.; Breutel, S.; Dumas, M.; ter Hofstede, A. (2007): Formal semantics and analysis of control flow in WS-BPEL. In: Science of Computer Programming 67 (2-3), S. 162–198.

[Ros06] Rosenkranz, Friedrich (2006): Geschäftsprozesse. Modell- und computergestützte Planung. 2., verb. Berlin: Springer.

[Sch01] Schöning, Uwe (2001): Theoretische Informatik - kurzgefasst. 4. Aufl. Heidelberg ; Berlin: Spektrum, Akad. Verl.

[Sta05] Stahl, Christian (2005): A Petri Net Semantics for BPEL. Informatik-Berichte 188. Humboldt-Universität zu Berlin.
Online verfügbar unter http://www2.informatik.hu-berlin.de/top/publikationen/en/Year/2005.complete.php#Stahl2005_hub_tr188

[TDG+07] Taylor, I.J; Deelman, E.; Gannon, D.B; Shields, M. (Hg.) (2007): Workflows for e-science. Scientific workflows for grids. London: Springer.

[WS04] W3C (2004): Web Services Architecture.
Online verfügbar unter http://www.w3.org/TR/ws-arch/

[WSDL07] W3C (2007): Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.
Online verfügbar unter http://www.w3.org/TR/wsdl20/

[XML08] W3C (2008): Extensible Markup Language (XML) 1.0 (Fifth Edition).
Online verfügbar unter http://www.w3.org/TR/xml/

[XSD04] W3C (2004): XML Schema Part 0: Primer Second Edition.
Online verfügbar unter http://www.w3.org/TR/xmlschema-0/

Alle in dieser Arbeit aufgeführten Weblinks wurden das letzte Mal am 25.08.2011 geprüft.

9 Verzeichnisse

9.1. Abbildungen

9.2. Regelmengen

9.3. Ableitungen

9.4. Listings

9.5. Definitionen

9.6. Tabellen

Erklärung

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

Unterschrift:

Stuttgart, den 31.08.2011