Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Diplomarbeit Nr. 3163

Ausführung von Grammatikbasierten Prozessmodellen in einer Cloud Umgebung

Florian Haupt

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.1.1, F.4.2, H.4.1

Inhaltsverzeichnis

1 Einleitung 5
1.1. Kapitelübersicht 6
1.2. Grundlagen 6
1.2.1. Workflow Technologie 6
1.2.2. BPEL 7
1.2.3. Web Services 7
1.2.4. Scientific Workflow 7
1.2.5. Cloud Computing 7
2 Formale Sprachen, Grammatiken, Automatenmodelle 9
2.1. Grundlagen 9
2.2. Klassifikation von Formalen Sprachen 12
2.2.1. Reguläre Sprachen (Chomsky Typ 3) 13
2.2.2. Kontextfreie Sprachen (Chomsky Typ 2) 15
2.2.3. Kontextsensitive Sprachen (Chomsky Typ 1) 17
2.2.4. Rekursiv aufzählbare Sprachen (Chomsky Typ 0) 18
3 Workflow Grammatiken (WoG) 20
3.1. Motivation 20
3.2. Grundlagen 21
3.3. Typen von Nichtterminalen 22
3.4. Produktionsregeln 23
3.5. Zusammenfassung 27
4 Anwendung von Grammatiken zur Erzeugung von Wörtern 28
4.1. Reguläre Grammatiken (Chomsky Typ 3) 31
4.2. Kontextfreie Grammatiken (Chomsky Typ 2) 33
4.3. Kontextsensitive Grammatiken (Chomsky Typ 1) 36
4.4. Rekursiv aufzählbare Grammatiken (Chomsky Typ 0) 38
4.5. Zusammenfassung 39
5 Aufbau einer Workflow Engine 40
5.1. Überblick über bestehende Architekturen 41
5.1.1. Workflow Management Coalition – The Workflow Reference Model 41
5.1.2. Apache ODE 43
5.1.3. Workflow Execution Engine (WEE) 44
5.1.4. Oracle BPEL Process Manager 45
5.1.5. YAWL Workflow System 46

5.2. Zusammenfassung der vorgestellten Workflow Management Systeme ... 48
5.3. Komponenten einer Workflow Engine ... 49
5.3.1. Navigator ... 49
5.3.2. Prozessmodellspeicher ... 49
5.3.3. Instanzspeicher ... 50
5.3.4. Auswertung von Ausdrücken ... 52
5.3.5. Programmaufrufkomponente ... 54
5.3.6. Erzeugen von Prozessinstanzen, Übergabeparameter ... 57
5.4. Ausführung von Workflows in einer Cloud Umgebung ... 58
5.4.1. Komponenten einer Workflow Engine in einer Cloud Umgebung ... 59
5.4.2. Anwendungsszenario 1 ... 61
5.4.3. Anwendungsszenario 2 ... 62
6 Entwurf einer Workflow Engine für die Ausführung von Workflow Grammatiken ... 64
6.1. Überführung von Workflow Grammatiken ein eine interne Repräsentation ... 64
6.1.1. Transformation in kontextfreie Grammatiken ... 65
6.1.2. Zuordnung von zu Regeln und Aktionen ... 66
6.1.3. Behandlung der generischen Regeln ... 71
6.1.4. Zusammenfassung ... 72
6.2. Aufbau einer Workflow Engine für Workflow Grammatiken ... 73
6.2.1. Navigator ... 73
6.2.2. Prozessmodellspeicher ... 73
6.2.3. Instanzspeicher ... 73
6.2.4. Prozessdatenspeicher und Auswertung von Ausdrücken ... 73
6.2.5. Programmaufruf ... 74
6.2.6. Schnittstellen nach außen ... 74
7 Prototyp einer Workflow Engine für Workflow Grammatiken ... 75
7.1. Dateiformat für Workflow Grammatiken ... 75
7.2. Interne Repräsentation für ausführbare Workflow Grammatiken ... 76
7.3. Funktionsweise des Navigators ... 76
7.4. Weitere Komponenten der Workflow Engine ... 79
7.5. Zusammenfassung ... 80
8 Zusammenfassung ... 81
9 Verzeichnisse ... 83

Abkürzungsverzeichnis

BPEL	Business Process Execution Language
BPM	Business Process Management
IAAS	Institut für Architektur von Anwendungssystemen
IaaS	Infrastructure as a Service
OaaS	Orchestration as a Service
ODE	Orchestration Director Engine
PaaS	Platform as a Service
QoS	Quality of Services
SaaS	Software as a Service
SEPL	Service Protocol Language
WEE	Workflow Execution Engine
WoG	Workflow Grammatik / Workflow Grammar
WS	Web Service
YAWL	Yet Another Workflow Language

1 Einleitung

Sowohl für die Beschreibung von Geschäftsprozessen als auch für die Beschreibung anderer Prozessarten, wie beispielsweise wissenschaftlicher Experimente (Scientific Workflows), existieren vielfältige Prozessmodelle und Prozessbeschreibungssprachen. Das Institut für Architektur von Anwendungssystemen (IAAS)[1] entwickelt eine Sprache zur Beschreibung von Prozessen, im Folgenden als *Workflow Grammatik (WoG)* bezeichnet, die auf dem Konzept der formalen Sprachen basiert. Ein Prozessmodell wird dabei durch eine Grammatik dargestellt. Eine Prozessausführung wird durch ein Wort der durch die Grammatik beschriebenen Sprache repräsentiert.

Workflow Grammatiken sind so aufgebaut, dass bestehende Prozessbeschreibungssprachen auf Workflow Grammatiken abbildbar sind. Damit wird erreicht, dass unterschiedliche Sprachen und Sprachtypen vergleichbar und sinnvoll klassifizierbar werden, indem man sie auf Workflow Grammatiken als Vergleichsbasis abbildet. Ein weiterer Aspekt, der im Zentrum dieser Arbeit steht, ist die einheitliche Ausführung. Prozessmodelle unterschiedlicher Sprachen können von der gleichen Workflow Engine ausgeführt werden, indem sie zunächst in Workflow Grammatik Modelle konvertiert und anschließend auf einer Workflow Grammatik Workflow Engine ausgeführt werden.

In dieser Arbeit wird die Ausführung von Workflow Grammatiken durch eine Workflow Engine untersucht. Eine Workflow Grammatik auszuführen bedeutet, mit Hilfe der Produktionsregeln einer solchen Grammatik ein Wort zu erzeugen. Aus der Theoretischen Informatik sind Automatenmodelle bekannt, die ebenfalls auf Grammatiken arbeiten. Diese Automaten produzieren jedoch keine Wörter, sie erkennen oder akzeptieren Wörter. Zunächst werden die prinzipiellen Unterschiede zwischen akzeptierenden und generierenden Automaten beschrieben und analysiert. Anschließend werden für die Grammatiktypen der Chomsky Hierarchie Verfahren zur Erzeugung von Wörtern entwickelt. Diese orientieren sich an den akzeptierenden Automatenmodellen, beachten jedoch zusätzlich die speziellen Eigenschaften von Workflow Grammatiken.

Ein weiterer Teil dieser Arbeit besteht in der Analyse des Aufbaus einer Workflow Engine. Nach einer kurzen Begriffsdefinition und -abgrenzung werden konkrete Beispielarchitekturen beschrieben und verglichen. Ausgehend von diesen Betrachtungen werden der allgemeine Aufbau und die prinzipielle Funktionsweise einer Workflow Engine beschrieben. Der dabei vorgestellte modulare Aufbau wird anschließend im Kontext der Ausführung von Workflows innerhalb einer Cloud Umgebung näher betrachtet. Anhand zweier Anwendungsszenarien wird gezeigt, wie die Modularisierung einer Workflow Engine dazu genutzt werden kann, einzelne Funktionalitäten in die Cloud Umgebung auszugliedern.

Die Betrachtungen sowohl zum Erzeugen von Wörtern aus einer Grammatik als auch zum allgemeinen Aufbau einer Workflow Engine werden anschließend kombiniert. Es wird der Aufbau einer modularen Workflow Engine zur Ausführung von Workflow Grammatiken vorgestellt. Dabei wird beschrieben, wie eine gegebene Workflow Grammatik in eine interne Repräsentation überführt wird, um diese anschließend effizient ausführen zu können. Basierend auf dem zuvor beschriebenen Entwurf wurde ein Prototyp einer Workflow Grammatik Workflow Engine entwickelt. Der Aufbau sowie die wesentlichen Aspekte der Implementierung des Prototypen werden abschließend kurz vorgestellt.

[1] http://www.iaas.uni-stuttgart.de/

1.1. Kapitelübersicht

Im Folgenden wird eine kurze Übereicht über die einzelnen Kapitel dieser Arbeit gegeben. Kapitel 1 beinhaltet eine allgemeine Einführung in das Thema dieser Arbeit, einen Überblick über ihren Aufbau sowie einige im Folgenden benötigten Grundlagen. Kapitel 2 gibt eine allgemeine Einführung in die Themenbereiche Formalen Sprachen, Grammatiken und Automatenmodelle. In Kapitel 3 wird eine weitere Grundlage dieser Arbeit, das Konzept der Workflow Grammatiken, vorgestellt und beschrieben.

Kapitel 4 stellt zu Beginn den Zusammenhang zwischen der Ausführung von Workflow Grammatiken und dem Erzeugen von Wörtern aus einer Grammatik her und beschreibt ihn detailliert. Anschließend werden, angelehnt an die in Kapitel 2 eingeführten Grammatikklassen, Verfahren zur Erzeugung von Wörtern aus einer Grammatik entwickelt. Dabei werden zum einen allgemeine Grammatiken, zum anderen aber auch Workflow Grammatiken und die damit verbundenen zusätzlichen Anforderungen an generierende Verfahren betrachtet.

In Kapitel 5 wird der zweite wesentliche Aspekt dieser Arbeit, der Aufbau einer Workflow Engine, betrachtet. Ausgehend von einer Analyse ausgewählter Architekturen werden die einzelnen Komponenten einer Workflow Engine allgemein beschrieben. Zum Schluss des Kapitels werden zwei Anwendungsszenarien für die Ausführung von Workflows in einer Cloud Umgebung vorgestellt.

In Kapitel 6 werden die in Kapitel 4 und 5 erarbeiteten Ergebnisse kombiniert und der Entwurf einer modularen Workflow Engine vorgestellt. Dabei wird ebenfalls gezeigt, wie die in Kapitel 3 vorgestellten Workflow Grammatiken für die Ausführung bearbeitet und in eine interne Repräsentation überführt werden. Kapitel 7 beschreibt die Umsetzung des im vorherigen Kapitel beschriebenen Entwurfs als Prototyp einer Workflow Grammatik Workflow Engine.

Der Inhalt dieser Arbeit wird abschließend in Kapitel 8 noch einmal zusammengefasst. Die wesentlichen Ergebnisse, die Einschränkungen dieser Arbeit sowie mögliche Folgearbeiten werden beschrieben. Kapitel 9 enthält das Literaturverzeichnis sowie weitere Verzeichnisse, wie beispielsweise das Abbildungsverzeichnis oder ein Verzeichnis aller verwendeten Abkürzungen.

1.2. Grundlagen

In den folgenden Abschnitten werden grundlegende Begriffe, Techniken und Technologien eingeführt, die im Rahmen dieser Arbeit verwendet oder referenziert werden. Es soll lediglich eine kurze Einführung in die genannten Themen gegeben werden. Falls im weiteren Verlauf dieser Arbeit nötig, werden einzelne Themen in den entsprechenden Kapiteln nochmals vertieft.

1.2.1. Workflow Technologie

Workflow Technologie behandelt die Abbildung von realen Geschäftsprozessen auf IT Umgebungen. Geschäftsprozesse können mit Hilfe von Prozessmodellen beschrieben werden. Sie spezifizieren den genauen Ablauf eines Geschäftsprozesses als Folge von Geschäftsfunktionen. Wird der Ablauf eines Geschäftsprozesses von einer IT Umgebung kontrolliert und bestehen die Geschäftsfunktionen aus Programmen, dann spricht man anstatt von Prozessmodellen und Prozessen von *Workflowmodellen* und *Workflows* [LR00].

1.2.2. BPEL

Die *Web Service Business Process Execution Language* (WS-BPEL, im Folgenden nur *BPEL*) ist eine XML basierte Sprache zur Beschreibung von Geschäftsprozessen. Geschäftsfunktionen sind in BPEL Prozessen als Web Services implementiert. Ein BPEL Prozess kann selbst wieder als Web Service zur Verfügung gestellt werden, weswegen BPEL auch als rekursive Aggregationssprache für Web Services bezeichnet wird. BPEL ist ein von der „Organization for the Advancement of Structured Information Standards“ (OASIS)[2] veröffentlichter Standard, die aktuelle Version ist WS-BPEL 2.0 [BPEL07].

1.2.3. Web Services

Als *Dienst* (*Service*) bezeichnet man Funktionen (Programme), welche über eine eindeutige Netzwerkadresse zur Verfügung gestellt werden. Dienste haben eine *„always on“ Semantik*, d.h. sie sind immer verfügbar.

Dienste, die auf Web Technologien wie beispielsweise HTTP oder XML basieren, bezeichnet man als *Web Services*. Die Menge aller Standards und Technologien im Zusammenhang mit Web Services bezeichnet man gesammelt auch als *Web Service Technologie* [WCL+05].

Die *Web Service Description Language (WSDL)* ist ein vom World Wide Web Consortium (W3C)[3] veröffentlichter Standard zur Beschreibung von Web Services. Er definiert ein Modell sowie eine XML Syntax, um die abstrakte Funktionalität sowie den Zugriff auf Web Services einheitlich zu beschreiben [WSDL07].

1.2.4. Scientific Workflow

Der Ursprung der Workflow Technologie liegt in der Automatisierung von Geschäftsprozessen. In diesem Bereich werden die entwickelten Technologien auch heute noch überwiegend eingesetzt. Workflow Technologien sind jedoch nicht auf diesen einen Anwendungsbereich beschränkt. Ein Beispiel für ein weiteres Einsatzgebiet ist die Wissenschaft.

Neben Theorie und Experiment hat sich der Bereich der Simulation und der Berechnung als drittes Standbein der Forschung etabliert [TDG+07]. Die Durchführung von computergestützten Berechnungen und Simulationen lässt sich ebenfalls als Prozess beschreiben. Daten müssen bewegt und bearbeitet werden, die Ausführung von Programmen oder Programmfunktionen muss koordiniert werden. In vielen Fällen handelt es sich dabei um Prozesse, welche in gleicher oder ähnlicher Form regelmäßig wiederholt werden. Diese Ausgangsbedingungen haben dazu geführt, dass Technologien aus dem Bereich der Automatisierung von Geschäftsprozessen in die Wissenschaftswelt übertragen wurden. Analog zum Begriff des *Business Workflow* spricht man in diesem Anwendungskontext auch von *Scientific Workflow*.

1.2.5. Cloud Computing

Der Begriff des *Cloud Computing* umschreibt ein Konzept zur Bereitstellung und Nutzung von IT Ressourcen. Die Grundidee des Cloud Computing besteht darin, IT Ressourcen als Dienst in einem Netzwerk zur Verfügung zu stellen. IT Ressourcen können dabei virtuelle Maschinen, Speicher oder auch Programme sein. Sie können innerhalb eines Intranets oder aber auch über das Internet zur Verfügung gestellt werden. Die Bereitstellung als Dienst bedeutet, dass diese Ressourcen jederzeit

[2] http://www.oasis-open.org/

[3] http://www.w3.org/

und nach Bedarf einfach genutzt werden können. Die Abrechnung von Cloud Computing Diensten erfolgt üblicherweise nutzungsabhängig [BKN11], [MG11].

Die durch Cloud Computing Dienste angebotenen Funktionalitäten erstrecken sich über eine große Bandbreite. Aus diesem Grund werden die heutzutage angebotenen Cloud Computing Lösungen üblicherweise in verschiedenen Kategorien eingeteilt. Eine solche ausführliche Klassifizierung wird in [LKN+09] vorgestellt, sie ist in Abbildung 1 vereinfacht dargestellt. Im Folgenden werden die wichtigsten Kategorien von Cloud Computing Diensten kurz beschrieben.

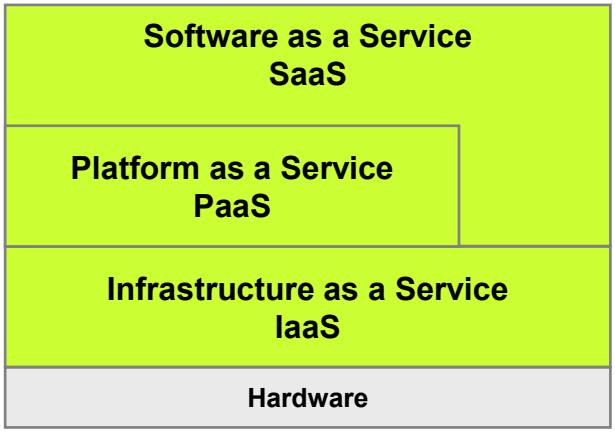


Abbildung 1: Cloud Computing

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) Dienste bieten Zugriff auf reale oder virtuelle Hardware. Neben der Bereitstellung virtueller Maschinen umfasst dies beispielsweise auch die Nutzung von Speicher- oder Netzwerkfunktionalitäten.

Platform as a Service (PaaS)

Platform as a Service (PaaS) Dienste bieten Ausführungsumgebungen an. Software kann entwickelt, in einer PaaS Cloud installiert und anschließend ausgeführt werden. Die angebotenen Plattform wird vom Cloud Anbieter verwaltet und skaliert automatisch. Die Realisierung eines PaaS Cloud Computing Dienstes kann auf IaaS Cloud Computing Diensten basieren.

Software as a Service (SaaS)

Software as a Service (SaaS) Dienste bieten Softwarefunktionalitäten als Dienst an. Die Software wird über ein Netzwerk angeboten und läuft in vielen Fällen in einem Web Browser. Die Installation und Wartung der angebotenen Software obliegt dem SaaS Anbieter. Die Realisierung eines SaaS Cloud Computing Dienstes kann auf PaaS Cloud Computing Diensten basieren.

2 Formale Sprachen, Grammatiken, Automatenmodelle

Im Mittelpunkt dieser Arbeit steht die Prozessbeschreibungssprache WoG (Workflow Grammatik). Der Aufbau und die Funktionsweise von Workflow Grammatiken werden in Kapitel 3 näher beschrieben. In Workflow Grammatiken wird ein Prozess nicht mit Hilfe eines Petrinetzes (wie beispielsweise in YAWL [AH05]) oder eines Graphen (wie beispielsweise in WSFL [WSFL01] oder teilweise auch in WS-BPEL [BPEL07]), sondern durch eine Grammatik beschrieben. Grammatiken werden unter anderem im Bereich der Theoretischen Informatik verwendet, sie werden aber auch in anderen Fachbereichen wie beispielsweise der Linguistik thematisiert [Mue09]. In der Theoretischen Informatik werden Grammatiken als ein mögliches Beschreibungsmittel für Formale Sprachen verwendet. Im Folgenden werden die wichtigsten Grundlagen zum Thema der Formalen Sprachen in der Theoretischen Informatik vorgestellt. Die verwendeten Bezeichnungen und Definitionen orientieren sich an [Sch08].

2.1. Grundlagen

Eine *Sprache* L ist definiert als eine (endliche oder unendliche) Menge von Wörtern. Ein *Wort* wiederum besteht aus *Zeichen* eines endlichen *Alphabets* Σ und besitzt eine endliche Länge. Die daraus folgende formale Definition einer Sprache lautet: $L \subseteq \Sigma^*$.

Endliche Sprachen (also endliche Mengen von Wörtern) können direkt aufgeschrieben werden. Die Sprache L_1, welche aus allen Wörtern der Länge zwei über dem Alphabet $\Sigma = \{a,b,c\}$ besteht, kann aufgeschrieben werden als $L_1 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$. Mit dieser Schreibweise ist L_1 vollständig und eindeutig beschrieben. Bei unendlichen Sprachen kann diese direkte Art der Beschreibung einer Sprache nicht mehr angewendet werden. Die unendliche Sprache L_2 über dem Alphabet $\Sigma = \{a,b,c\}$, welche alle Wörter enthält, die nur aus dem Zeichen a bestehen, kann nicht direkt als Menge aufgeschrieben werden. Die Schreibweise $L_2 = \{a, aa, aaa, ...\}$ ist nicht eindeutig, sie ist abhängig von der Interpretation von „…“. Allgemein benötigt man also eine Möglichkeit, unendliche Mengen mit Hilfe eines endlichen Beschreibungsmittels eindeutig und vollständig zu beschreiben.

Grammatiken sind ein solches Beschreibungsmittel. Sie bestehen im Wesentlichen aus einem Regelwerk. Durch ein- oder mehrmalige Anwendung dieser Regeln können Wörter produziert werden. Das Erzeugen von Wörtern mit Hilfe einer Grammatik wird in Kapitel 4 ausführlich diskutiert. Eine Grammatik G_i „gehört“ zu einer Sprache L_i, wenn durch die Anwendung der Regeln von G_i genau die Menge der Wörter aus L_i erzeugt wird. Man schreibt dies als $L_i = L(G_i)$ und sagt „L_i ist die von G_i erzeugte Sprache“. In Definition 1 ist der allgemeine Aufbau einer Grammatik dargestellt.

G = (V, E, P, S)

V = Menge der Nichtterminale (auch: Variablen)

E = Menge der Terminale (auch: Alphabet)

P = Menge der Produktionsregeln, $P \subset (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$

S = das Startsymbol, muss in V enthalten sein

Definition 1: Formale Grammatik

Um mit Hilfe einer Grammatik Wörter zu erzeugen, werden die *Produktionsregeln* einer Grammatik wiederholt angewendet. Die Menge aller Regelanwendungen zur Produktion eines Wortes bezeichnet man auch als *Ableitung*. Eine Ableitung beginnt immer mit dem *Startsymbol* und der Anwendung einer passenden Produktionsregel auf das Startsymbol. Daraus entsteht im Allgemeinen ein sogenanntes *Teilwort*. Ein Teilwort besteht sowohl aus Nichtterminalen als auch aus Terminalen. Ein Wort besteht,

im Gegensatz zu einem Teilwort, ausschließlich aus Terminalen. Auf ein Teilwort können erneut Produktionsregeln angewendet werden. Das wiederholte Anwenden von Produktionsregeln auf Teilworte wird so lange wiederholt, bis keine Produktionsregel mehr angewendet werden kann oder ein Wort erzeugt wurde. Besteht das letzte erzeugt Teilwort nur noch aus Terminalen, so ist es ein Wort der Grammatik, die Ableitung war erfolgreich. Eine Ableitung eines Wortes ist also eine Abfolge von Teilwörtern, die mit dem Startsymbol beginnt und mit dem entsprechenden Wort endet.

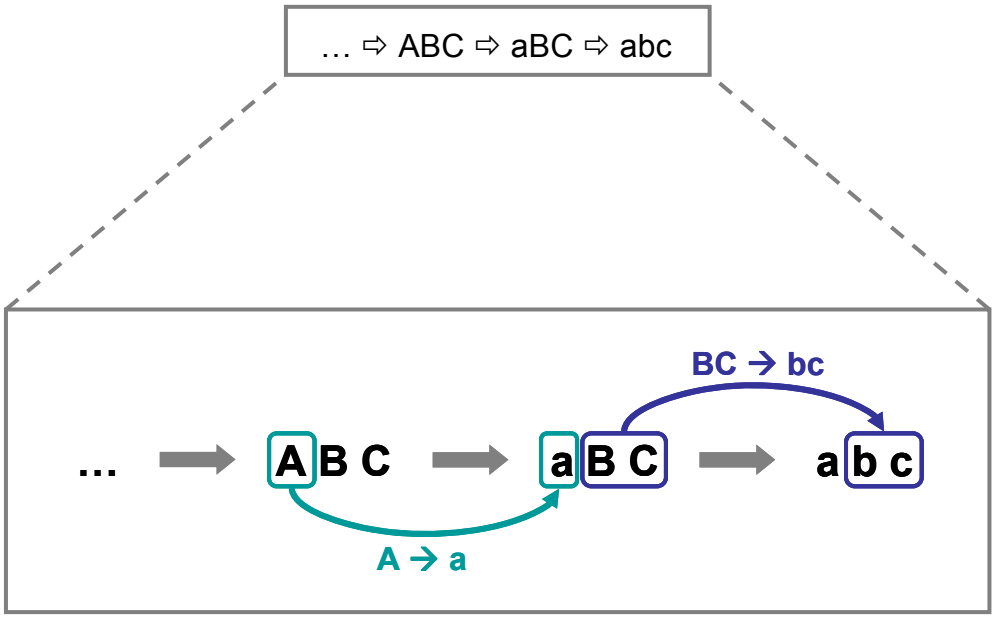


Abbildung 2: Anwendung von Produktionsregeln

Eine Produktionsregel beschreibt eine Ersetzung innerhalb eines Teilwortes[4]. Sie kann nur dann auf ein Teilwort angewendet werden, wenn die linke Seite der Regel in dem Teilwort enthalten ist. Die Anwendung der Regel bedeutet, dass genau ein Vorkommen der linken Seite durch die rechte Seite ersetzt wird. Eine solche Ersetzung ist eine atomare Aktion. Abbildung 2 stellt die Anwendung von Produktionsregeln an einem Beispiel dar. Im ersten Schritt wird das Nichtterminal „A“ durch das Terminal „a“ ersetzt. Im zweiten Schritt wird das Teilwort „BC“ durch das Teilwort „bc“ ersetzt.

In Abbildung 3 ist die Erzeugung des Wortes „abc“ anhand einer einfachen Beispielgrammatik dargestellt. Es sind drei mögliche Ableitungen für dieses Wort dargestellt. Unter den Ableitungen sind die dazugehörigen sogenannten *Ableitungsbäume* dargestellt. Ableitungsbäume beschreiben ebenfalls die Ableitung eines Wortes, sie machen aber im Allgemeinen keine eindeutigen Aussagen über die Reihenfolge der Anwendung der einzelnen Regeln. Die Wurzel eines Ableitungsbaumes ist immer das Startsymbol, die Blätter beschreiben das erzeugte Wort.

Zu jedem durch eine Grammatik erzeugbaren Wort kann es mehrere Ableitungen geben. Ableitungen können auf Ableitungsbäume abgebildet werden, dabei ist es möglich, dass mehrere Ableitungen auf den gleichen Ableitungsbaum abgebildet werden. Die in Abbildung 3 dargestellte Ableitung (1) wendet die gleichen Regeln wie Ableitung (3) an, die beiden Ableitungen unterscheiden sich lediglich in der Reihenfolge der Regelanwendungen. Sie werden in diesem Beispiel beide auf den linken Ableitungsbaum abgebildet. Dies verdeutlicht, dass ein Ableitungsbaum mehrere Ableitungen

[4] Grammatiken sind eine spezielle Form sogenannter Semi-Thue Systeme, die auch als Wortersetzungssysteme bezeichnet werden [Tho10]. Im Gegensatz zu Grammatiken gibt es in Semi-Thue Systemen beispielsweise kein explizites Startsymbol und keine Unterscheidung in Terminale und Nichtterminale.

beschreiben kann. Die Ableitung (2) verwendet dagegen andere Regeln als die Ableitungen (1) und (3), demzufolge wird diese Ableitung auf den rechten Ableitungsbaum abgebildet. Eine Grammatik, in der es für ein Wort mehr als einen Ableitungsbaum gibt, nennt man *mehrdeutig*.

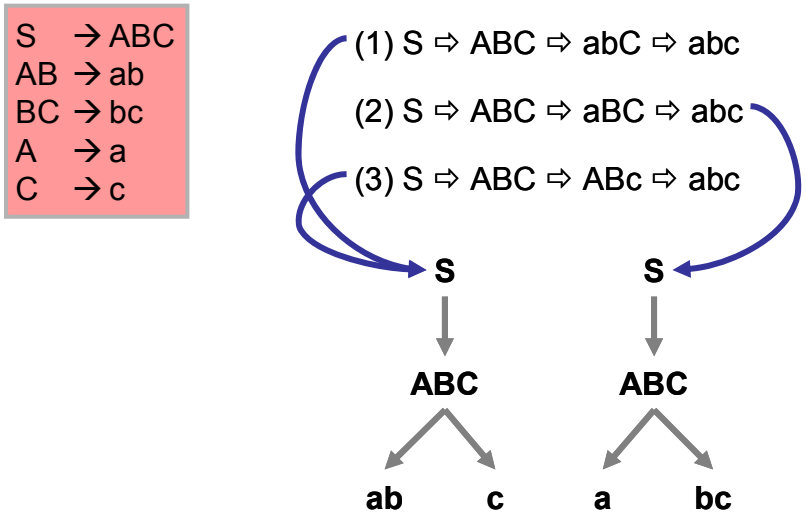


Abbildung 3: Grammatik, Ableitung und Ableitungsbaum

Eine zweite Möglichkeit, Sprachen (und insbesondere auch unendliche Sprachen) zu beschreiben, sind *Automaten*. Die unterschiedlichen Automatenmodelle, die zur Beschreibung formaler Sprachen verwendet werden, bezeichnet man auch als Akzeptoren [WH09]. Dieser Begriff leitet sich aus ihrer Funktionsweise ab.

Ein Automat (im Sinne der formalen Sprachen) bekommt ein Wort übergeben und beginnt dann zu „arbeiten“. Der Automat kann seine Arbeit in endlicher Zeit beenden oder aber auch unendlich lange laufen. Wenn er seine Arbeit beendet, dann akzeptiert er das Wort oder er akzeptiert es nicht. Wie genau dieses Akzeptieren oder nicht Akzeptieren geschieht, ist unterschiedlich und abhängig vom Automatenmodell und seiner formalen Definition. Das Akzeptieren oder nicht Akzeptieren kann beispielsweise durch den Zustand signalisiert werden, in dem sich der Automat nach seiner Terminierung befindet. Es ist aber auch möglich, das Akzeptieren oder nicht Akzeptieren durch eine explizite Ausgabe zu zeigen.

Automaten bekommen ihre Eingabe auf einem *Eingabeband* übergeben. Sie können, beginnende beim ersten Zeichen, die gesamte Eingabe schrittweise von links nach rechts einlesen. Ist die Eingabe einmal eingelesen, ist keine weitere Operation auf dem Eingabeband mehr möglich. Ähnlich zum Eingabeband besitzen einige Automatenmodelle ein *Ausgabeband*. Diese Band ist zu Beginn leer. Es kann zeichenweise von links nach rechts beschrieben werden. Vom Ausgabeband kann nicht gelesen werden. Geschriebene Zeichen können nicht verändert oder überschrieben werden. Neben Eingabe- und Ausgabeband besitzen die mächtigeren Automatenmodelle, wie beispielsweise Turingmaschinen, ein *Arbeitsband*. Auf dem Arbeitsband kann beliebig gelesen und geschrieben werden. Automaten können sich auf dem Eingabeband frei bewegen.

Die zu einem Automaten M_i gehörende Sprache L_i ist die Menge aller Wörter, die dieser Automat akzeptiert. Man sagt auch „L_i ist die von M_i akzeptierte Sprache“ und schreibt $L_i = L(M_i)$. Der Aufbau eines Automaten hängt vom verwendeten Automatenmodell ab. Die einzelnen Modelle und ihre formale Definition werden im folgenden Kapitel beschrieben. Vorausgreifend sei aber hier schon festgehalten, dass die Definition eines Automaten, ebenso wie die einer Grammatik, endlich ist. Damit bilden Automaten ein weiteres endliches Beschreibungsmittel für unendliche Sprachen.

2.2. Klassifikation von Formalen Sprachen

In der Theoretischen Informatik unterscheidet man verschiedenen Typen, oder auch Klassen, von Sprachen. Die bekannteste und grundlegendste Klassifizierung ist die Chomsky-Hierarchie. Sie ist in Abbildung 4 dargestellt. Die Menge aller Sprachen enthält dabei auch die Sprachen, die nicht durch Grammatiken darstellbar sind. Die Menge der Typ 0 Sprachen entspricht der Menge der Sprachen, die durch Grammatiken darstellbar sind. Die Chomsky Hierarchie ist eine Untermengenkonstruktion, das bedeutet, dass jede Sprache vom Typ x immer auch eine Sprache vom Typ x-1 ist (beispielsweise ist jede reguläre Sprache immer auch kontextfrei).

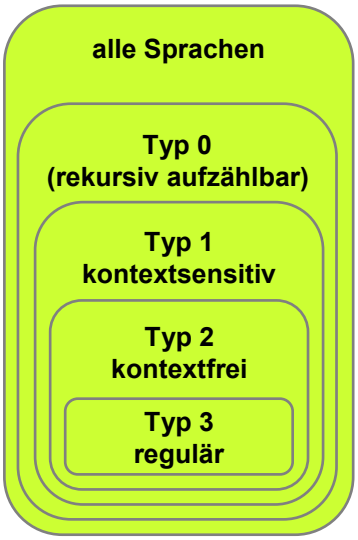


Abbildung 4: Chomsky Hierarchie

Die Definition der Chomsky Hierarchie erfolgt über Grammatiken, sie klassifiziert also im Grunde keine Sprachen, sondern Grammatiken. Im vorherigen Kapitel wurde jedoch bereits ausführlich beschrieben, dass jede Grammatik immer auch eine Sprache beschreibt. Demzufolge kann die Chomsky Hierarchie sowohl auf Grammatiken als auch auf Sprachen angewendet werden. Ein ähnlicher Zusammenhang besteht, wie ebenfalls bereits beschrieben, zwischen Sprachen und Automaten. Zu den Sprachen jedes Chomsky Typs gibt es immer auch ein Automatenmodell, welches genau die Sprachen dieses Typs erkennen kann.

Die den einzelnen Chomsky Typen zugeordneten Automatenmodelle unterscheiden sich sowohl ihrer Mächtigkeit als auch in ihrem Laufzeitverhalten. Ein Endlicher Automat kann beispielsweise nur Typ 3 Sprachen erkennen, dafür tut er dies in linearer Zeit. Ein Kellerautomat kann dagegen zusätzlich auch Typ 2 Sprachen erkennen, dafür kann er dies aber im allgemeinen Fall nicht in linearer Zeit tun.

Die Klassifizierung von Sprachen nach der Chomsky Hierarchie erlaubt es, detaillierte Aussagen über ihre Eigenschaften zu machen. Im Folgenden werden ein paar Beispiele genannt. Ist eine Sprache regulär, dann ist diese Sprache in linearer Zeit entscheidbar. Ist sie dagegen kontextsensitiv, dann ist sie weiterhin entscheidbar, aber nicht mehr in linearer Zeit. Das Äquivalenzproblem („Erzeugen zwei Grammatiken die gleiche Sprache?“) ist für Typ 3 Grammatiken entscheidbar, für Typ 2 Grammatiken dagegen bereits nicht mehr.

In den folgenden Unterkapiteln werden die Sprachklassen gemäß der Chomsky Hierarchie beschrieben. Zu jeder Sprachklasse werden der entsprechende Grammatiktyp sowie das dazugehörige Automatenmodell vorgestellt und formal definiert.

2.2.1. Reguläre Sprachen (Chomsky Typ 3)

Reguläre Sprachen stellen in der Chomsky Hierarchie die kleinste Sprachklasse dar. Reguläre Sprachen können durch reguläre Grammatiken oder durch reguläre Ausdrücke beschrieben werden. Reguläre Sprachen werden durch (sowohl deterministische als auch nichtdeterministische) endliche Automaten erkannt.

2.2.1.1. Reguläre Grammatiken

Reguläre Sprachen können durch reguläre Grammatiken beschrieben werden. Eine Grammatik ist regulär, wenn ihre Regeln den folgenden Einschränkungen gehorchen.

- Die linke Seite einer Regel muss aus genau einem Nichtterminal bestehen.
- Die rechte Seite einer Regel muss entweder aus genau einem Terminal oder aus einem Terminal gefolgt von einem Nichtterminal aufgebaut sein.

Eine Grammatik mit derartigen Regeln nennt man auch „rechtsregulär“ (die Anwendung der Regeln lässt ein Wort nach rechts wachsen). Eine alternative und äquivalente Definition einer regulären Grammatik fordert für die rechte Seite einer Regel, dass sie aus einem Nichtterminal gefolgt von einem Terminal besteht. Eine so definierte Grammatik nennt man „linksregulär“. In dieser Arbeit wird der Begriff „regulär“ als Synonym für „rechtsregulär“ verwendet.

```
G1 = (N1, T1, P1, S), N1 = {S, B}, T1 = {0, 1}
P1 = { S → 0,
       S → 1,
       S → 1B,
       B → 0B,
       B → 1B,
       B → 0,
       B → 1}
```

Listing 1: Beispiel für eine rechtsreguläre Grammatik

Ein Beispiel für eine rechtsreguläre Grammatik ist in Listing 1 dargestellt. Die Grammatik G_1 beschreibt die Menge aller Binärzahlen ohne führende Nullen. Die gleiche Sprache wird von der in Listing 2 dargestellten linksregulären Grammatik G_2 erzeugt. Formal ausgedrückt gilt also $L(G_1) = L(G_2)$.

```
G2 = (N2, T2, P2, S), N2 = {S, B}, T2 = {0, 1}
P2 = { S → 0,
       S → 1,
       S → B0,
       S → B1,
       B → B0,
       B → B1,
       B → 1}
```

Listing 2: Beispiel für eine linksreguläre Grammatik

2.2.1.2. Reguläre Ausdrücke

Eine weitere Beschreibungsmöglichkeit für reguläre Sprachen sind reguläre Ausdrücke. Reguläre Ausdrücke beschreiben Wortmengen durch eine Art Muster und sind aufgrund ihrer kompakten Darstellung in der Praxis weit verbreitet.

In Definition 2 werden reguläre Ausdrücke über dem Alphabet Σ formal definiert. Die Regeln (1) und (2) definieren zunächst atomare reguläre Ausdrücke. Die Regeln (3) und (4) erweitern die Definition regulärer Ausdrücke rekursiv.

(1) ε	Das leere Wort ist ein regulärer Ausdruck.
(2) a	Jedes Zeichen des Alphabets Σ ist ein regulärer Ausdruck.
(3) a|b	Sind a und b reguläre Ausdrücke, dann ist auch ihre Vereinigung ein regulärer Ausdruck.
(4) a*	Ist a ein regulärer Ausdruck, so sind auch beliebige Konkatenationen von a ein regulärer Ausdruck (Kleene Operator, endlicher Abschluss). $a^0 = \varepsilon$ $a^3 = aaa$

Definition 2: Reguläre Ausdrücke

Ein Beispiel für einen regulären Ausdruck ist in Listing 3 dargestellt. Der reguläre Ausdruck BIN beschreibt die gleiche Sprache wie die in Listing 1 und Listing 2 aufgeführten Grammatiken, also die Menge aller Binärzahlen ohne führende Nullen. Es gilt $L(BIN) = L(G_1) = L(G_2)$.

```
BIN = 0 | 1(0|1)*
```

Listing 3: Beispiel für einen regulären Ausdruck

2.2.1.3. Endliche Automaten

Ein endlicher Automat liest ein Wort zeichenweise ein. Abhängig von den eingelesenen Zeichen navigiert er durch einen Zustandsgraphen. Ist das Wort komplett eingelesen und der Automat befindet sich ein einem Endzustand, so akzeptiert er das Wort. Befindet er sich dagegen in einem Zustand, der kein Endzustand ist, akzeptiert er das Wort nicht.

$DEA = (Z, \Sigma, \delta, z_0, E)$

Z = Menge der Zustände

Σ = Eingabealphabet, $Z \cap \Sigma = 0$

δ = Überführungsfunktion, $\delta : Z \times \Sigma \to Z$

z_0 = Startzustand, $z_0 \in Z$

E = Menge der Endzustände, $E \subseteq Z$

Definition 3: Endlicher Automat

Definition 3 beschreibt einen Deterministischen Endlichen Automaten formal. Die Überführungsfunktion δ beschreibt das Verhalten des Automaten. Sie definiert die Zustandsübergänge des Automaten abhängig von den eingelesenen Zeichen. Der wesentliche Unterschied zwischen einem Deterministischen Endlichen Automaten (DEA) und einem Nichtdeterministischem Endlichen Automaten (NEA) besteht in den Zustandsübergängen. Ein DEA darf, ausgehend von einem beliebigen Zustand, für jedes mögliche Zeichen höchstens einen Übergang besitzen. Ein NEA unterliegt dieser Beschränkung nicht. Die Definition der Überführungsfunktion eines

Nichtdeterministischen Endlichen Automaten lautet dann $\delta : Z \times \Sigma \to \mathrm{P}(Z)$, P bezeichnet dabei die Potenzmenge.

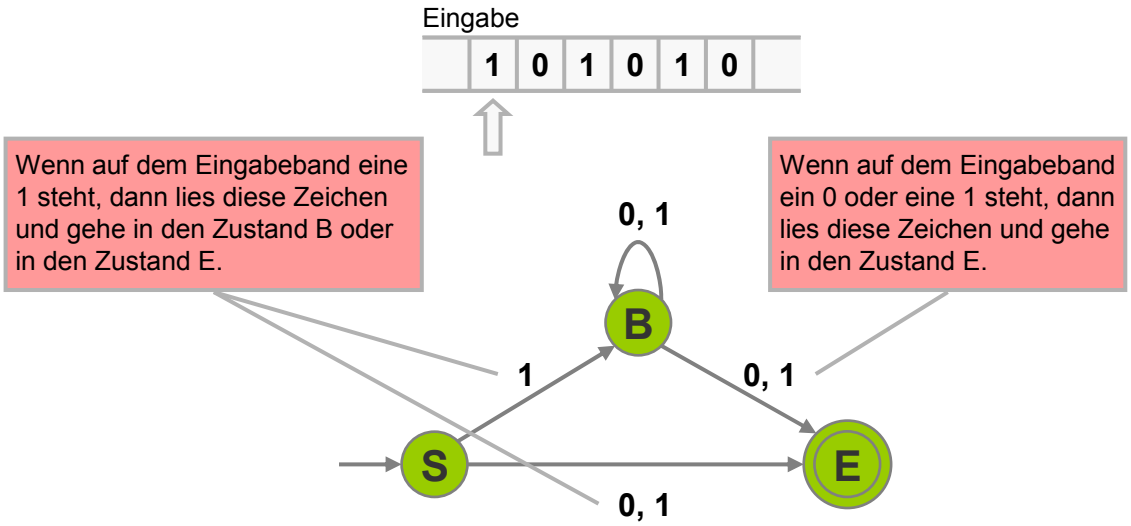


Abbildung 5: Endlicher Automat, nichtdeterministisch (NEA)

Abbildung 5 stellt den NEA M_1 dar, der genau die Sprache aller Binärzahlen ohne führende Nullen akzeptiert. Der in Abbildung 6 dargestellte DEA M_2 erkennt die gleiche Sprache wie M_1. Damit gilt $L(M_1) = L(M_2) = L(BIN) = L(G_1) = L(G_2)$.

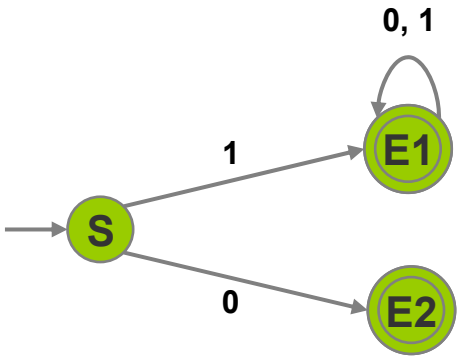


Abbildung 6: Endlicher Automat, deterministisch (DEA)

2.2.2. Kontextfreie Sprachen (Chomsky Typ 2)

Kontextfreie Sprachen stellen in der Chomsky Hierarchie die nächst höhere Sprachklasse oberhalb der regulären Sprachen dar. Kontextfreie Sprachen können durch kontextfreie Grammatiken beschrieben werden. Kontextfreie Sprachen werden durch (nichtdeterministische) Kellerautomaten erkannt.

2.2.2.1. Kontextfreie Grammatiken

Kontextfreie Sprachen können durch *kontextfreie Grammatiken* beschrieben werden. Eine Grammatik ist kontextfrei, wenn ihre Regeln den folgenden Einschränkungen gehorchen.

- Die linke Seite einer Regel muss aus genau einem Nichtterminal bestehen.
- Die rechte Seite einer Regel muss mindestens ein Terminal oder Nichtterminal enthalten, ansonsten ist sie keinerlei Einschränkungen unterworfen.

```
G3 = (N3, T3, P3, S), N3 = {S }, T3 = {a, b}
P3 = { S → ab,
       S → aSb}
```

Listing 4: Beispiel für eine kontextfreie Grammatik

Ein einfaches Beispiel für eine kontextfreie Grammatik ist in Listing 4 dargestellt. Die Grammatik G_3 beschreibt die Menge aller Wörter die aus einer Menge von „a“ gefolgt von der gleichen Anzahl von „b“ bestehen, also beispielsweise „ab“ oder „aaabbb“.

2.2.2.2. Kellerautomaten

Ein *Kellerautomat* besitzt, im Gegensatz zu Endlichen Automaten, einen Speicher. Dieser Speicher ist als Keller organisiert (FIFO: First In, First Out), es kann immer nur das oberste Element gelesen werden. Ein Kellerautomat kann in jedem Schritt ein oder kein Zeichen vom Eingabeband einlesen. Abhängig vom eingelesenen Zeichen und dem obersten Element des Kellers navigiert er durch einen Zustandsgraphen. Bei jedem Übergang können beliebig viele Elemente im Kellerspeicher abgelegt werden. Ein Kellerautomat besitzt (in der in dieser Arbeit verwendeten Definition) keine Endzustände. Ein Wort wird genau dann akzeptiert, wenn es komplett eingelesen ist und der Kellerspeicher anschließend leer ist. Ist der Kellerspeicher nach dem Einlesen des Wortes nicht leer, so wird dieses Wort nicht akzeptiert.

Kellerautomat = $(Z, \Sigma, \Gamma, \delta, z_0, \#)$

Z = Menge der Zustände

Σ = Eingabealphabet

Γ = Kelleralphabet

δ = Überführungsfunktion, $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to P_e(Z \times \Gamma^*)$, P_e bezeichnet dabei alle endlichen Teilmengen

z_0 = Startzustand, $z_0 \in Z$

$\#$ = das unterste Kellerzeichen, $\# \subseteq \Gamma$, repräsentiert den leeren Keller

Definition 4: (Nichtdeterministischer) Kellerautomat

Definition 4 beschreibt einen *Nichtdeterministischen Kellerautomaten*. Wir im Folgenden von Kellerautomaten gesprochen, dann sind immer Nichtdeterministische Kellerautomaten gemeint. *Deterministische Kellerautomaten* sind nicht äquivalent zu Nichtdeterministischen Kellerautomaten. Deterministische Kellerautomaten können genau die Menge der *deterministisch kontextfreien Sprachen* erkennen. Diese Menge ist eine echte Untermenge der kontextfreien Sprachen.

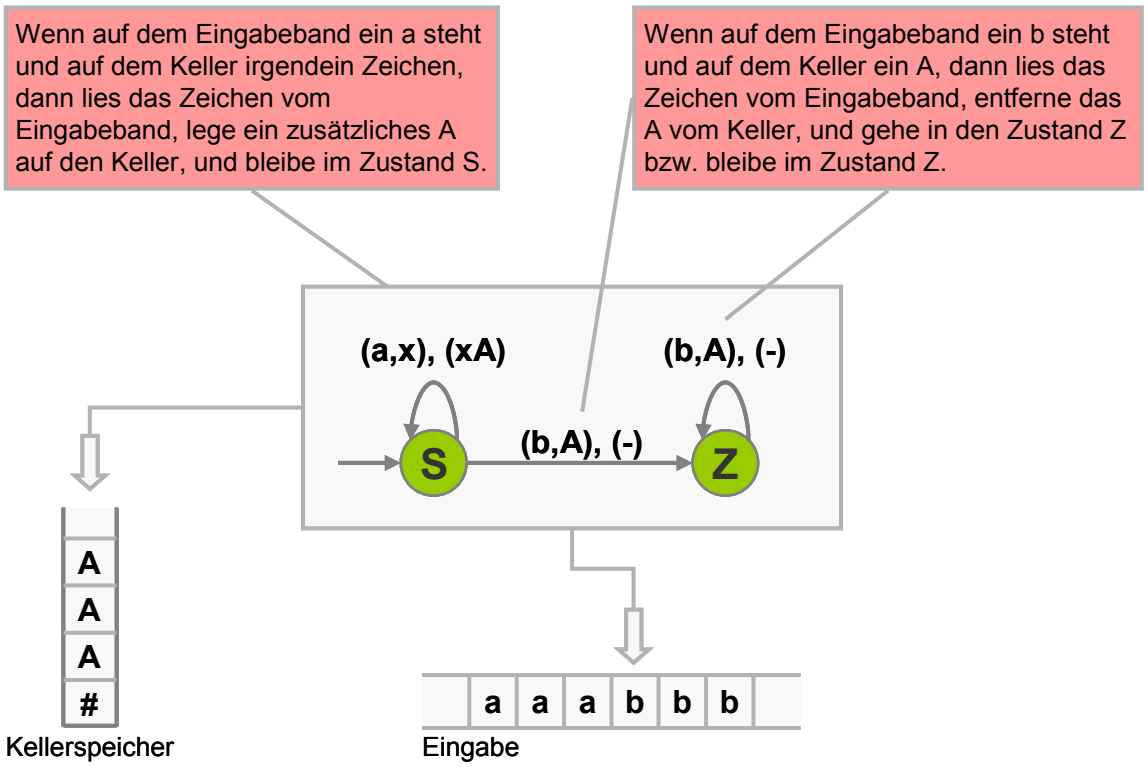


Abbildung 7: (Nichtdeterministischer) Kellerautomat

Abbildung 7 stellt einen Kellerautomaten dar, der genau die Sprache erkennt, welche durch die in Listing 4 definierte Grammatik beschrieben wird. Für jedes gelesene „a“ wird ein „A“ auf dem Keller abgelegt. Für jedes anschließend eingelesene „b“ wird wieder ein „A“ entfernt. Enthält das eingelesene Wort also gleiche viele „a“ und „b“, dann ist der Keller am Ende wieder leer, das Wort wird also akzeptiert. Durch den nur einmal möglichen Übergang vom Zustand S in den Zustand Z wird zusätzlich sichergestellt, dass zuerst nur „a“ und danach nur „b“ eingelesen werden können.

2.2.3. Kontextsensitive Sprachen (Chomsky Typ 1)

Kontextsensitive Sprachen stellen in der Chomsky Hierarchie die nächst höhere Sprachklasse oberhalb der kontextfreien Sprachen dar. Kontextsensitive Sprachen können durch kontextsensitive Grammatiken beschrieben werden. Kontextsensitive Sprachen werden durch linear beschränkt Turingmaschinen (LBA, Linear Bounded Automaton) erkannt.

2.2.3.1. Kontextsensitive Grammatiken

Kontextsensitive Sprachen können durch *kontextsensitive Grammatiken* beschrieben werden. Eine Grammatik ist kontextsensitiv, wenn ihre Regeln der folgenden Einschränkung gehorcht.

- Die rechte Seite einer Regel muss mindestens so viele Elemente wie die linke Seite enthalten (sie muss mindestens gleich lang oder länger sein), ansonsten ist sie keinerlei Einschränkungen unterworfen.

Das bedeutet, dass die Teilwörter einer Ableitung niemals kleiner werden dürfen. Sie müssen immer wachsen oder zumindest gleich groß bleiben. Diese Eigenschaft stellt die Entscheidbarkeit von Typ 1 Sprachen sicher.

```
G4 = (N4, T4, P4, S), N4 = {S, B, C, H}, T4 = {a, b, c}

P3 = { S  → aSBC,
       S  → aBC,
       CB → HB,
       HB → HC,
       HC → BC,
       aB → ab,
       bB → bb,
       bC → bc,
       cC → cc}
```

Listing 5: Beispiel für eine kontextsensitive Grammatik

Ein Beispiel für eine kontextsensitive Grammatik ist in Listing 5 dargestellt. Die Grammatik G_4 beschreibt die Menge aller Wörter die aus einer Menge von „a“ gefolgt von der gleichen Anzahl von „b“ und „c“ bestehen, also beispielsweise „abc“ oder „aaabbbccc“.

2.2.3.2. Linear beschränkte Turingmaschinen (LBA)

Linear beschränkte Turingmaschinen (LBA) sind allgemeine *Turing Maschinen (TM)*, die den Bereich des Arbeitsbandes, auf dem das Eingabewort steht, während ihrer Ausführung nie verlassen. Der Aufbau und die Arbeitsweise allgemeiner Turing Maschinen wird im Kapitel 2.2.4.2 kurz beschrieben.

2.2.4. Rekursiv aufzählbare Sprachen (Chomsky Typ 0)

Rekursiv aufzählbare (semi-entscheidbare) Sprachen stellen in der Chomsky Hierarchie die nächst höhere Sprachklasse oberhalb der kontextfreien Sprachen dar. Rekursiv aufzählbare Sprachen können durch Typ 0 Grammatiken beschrieben werden. Rekursiv aufzählbare Sprachen werden durch Turingmaschinen (TM) erkannt.

2.2.4.1. Typ 0 Grammatiken

Typ 0 Sprachen können durch *allgemeine formale Grammatiken* beschrieben werden. Eine solche Grammatik unterliegt keinerlei Einschränkungen, ihr Aufbau wird durch die bereits vorgestellte Definition 1: Formale Grammatik beschrieben.

2.2.4.2. Turingmaschinen (TM)

Die bisher vorgestellten Automatenmodelle bekommen ein Wort übergeben und können dieses zeichenweise einlesen. Sie besitzen, wenn überhaupt, nur eingeschränkte Speichermöglichkeiten. Endliche Automaten besitzen keinerlei Speicher, Kellerautomaten besitzen einen eingeschränkten Speicher, den Kellerspeicher. Eine *Turingmaschine (TM)* dagegen bekommt das Eingabewort auf einem sogenannten *Band* übergeben. Die Größe des Bandes ist im Allgemeinen unbeschränkt. Sie kann sich auf diesem Band frei bewegen (in jedem Schritt eine Position nach links oder rechts) und dabei einzelne Zeichen sowohl lesen als auch schreiben. Das Verhalten einer Turingmaschine wird, ähnlich wie bei den anderen Automatenmodellen, über eine Überführungsfunktion definiert. Abhängig vom aktuellen Zustand und dem Zeichen auf dem Band kann der Zustand gewechselt, ein Zeichen auf das Band geschrieben und die Position auf dem Band verändert werden.

Kellerautomat = $(Z, \Sigma, \Gamma, \delta, z_0, \#, E)$

Z = Menge der Zustände

Σ = Eingabealphabet

Γ = Bandalphabet, $\Gamma \supset \Sigma$

δ = Überführungsfunktion, $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$, $\{L,R,N\}$ bezeichnet dabei die möglichen Bewegungen „links", „rechts" und „stehen bleiben"

z_0 = Startzustand, $z_0 \in Z$

b = das Blank, repräsentiert ein leeres Feld auf dem Band, $b \in \Gamma - \Sigma$

E = Menge der Endzustände, $E \subseteq Z$

Definition 5: Turingmaschine

Turingmaschinen werden als das mächtigste aller Automatenmodelle angesehen. Diese Auffassung wird in der Churchschen These formuliert, sie ist in Definition 6 dargestellt. Die Churchsche These besagt, dass alles, was im menschlich intuitiven Sinne berechenbar ist, mit einer Turingmaschine berechenbar ist. Das Modell der Turingmaschine ist damit ein universelles Automatenmodell.

Die durch die formale Definition der Turing-Berechenbarkeit erfasste Klasse von Funktionen stimmt genau mit der Klasse der im intuitiven Sinne berechenbaren Funktionen überein.

Definition 6: Churchsche These (nach [Sch08])

3 Workflow Grammatiken (WoG)

Workflow Grammatiken als Beschreibungsmittel für ausführbare Prozesse wurden in [Vuk11] entwickelt und ausführlich beschrieben. Workflow Grammatiken sind formale Grammatiken, die als Prozessmodelle interpretiert werden können. Viele Workflowsprachen sind graphbasiert. Ein Prozessmodell wird mit Hilfe von Knoten und Kanten modelliert. Die Knoten stellen Aktivitäten dar, die Kanten definieren Kontroll- oder Datenfluss. Workflow Grammatiken dagegen sind formale Grammatiken, die mit einer zusätzlichen Semantik angereichert werden. Nichtterminale können laufende Aktivitäten oder Variablen repräsentieren, Terminale stehen für beendete Aktivitäten. Eine Produktionsregel einer Workflow Grammatik beschreibt eine Aktion, die während der Ausführung des durch die Workflow Grammatik beschriebenen Prozessmodells durchgeführt werden kann. Eine Ableitung entspricht der Ausführung eines Prozesses, ein Wort beschreibt eine beendete Prozessinstanz.

Die für die Ausführung von Workflow Grammatiken wesentlichen Aspekte werden in Kapitel 4 detailliert behandelt. Im Folgenden wird zunächst der Entwurf von Workflow Grammatiken kurz motiviert, anschließend werden die einzelnen Konstrukte zur Modellierung eines Prozess mit Hilfe einer Grammatik vorgestellt. Dieses Kapitel stellt damit im Wesentlichen eine Zusammenfassung von [Vuk11] mit dem Fokus auf die in dieser Arbeit relevanten Aspekte von Workflow Grammatiken dar.

3.1. Motivation

Es existieren viele unterschiedliche Sprachen für die Modellierung von Prozessen. Sie unterscheiden sich in ihren Fähigkeiten, ihren bevorzugten Anwendungsbereichen und können völlig unterschiedliche Paradigmen verfolgen. Der Entwurf von Workflow Grammatiken verfolgt das Ziel, dass alle diese Workflow Sprachen auf Workflow Grammatiken abgebildet werden können. Dies bedeutet, dass zu jedem beliebigen Prozessmodell eine (Ausführungs-) äquivalente Workflow Grammatik angegeben werden kann. Diese Eigenschaft bringt zwei große Vorteile mit sich, die hier kurz dargestellt werden sollen.

Prozessmodelle unterschiedlicher Modellierungssprachen sind in vielen Fällen schwer zu vergleichen. Es können unterschiedliche Modellierungsparadigmen benutzt werden, die gleichen (gleich benannten) Konstrukte können unterschiedliche Semantiken besitzen. Bildet man diese Prozessmodelle jedoch auf Workflow Grammatiken ab, dann können sie auf dieser einheitlichen Basis sinnvoll verglichen werden. Ebenso möglich ist eine Klassifizierung einzelner Modellierungssprachen, indem ihre Abbildung auf Workflow Grammatiken allgemein beschrieben und anschließend analysiert wird.

Der zweite Vorteil der Verwendung von Workflow Grammatiken beruht darauf, dass unterschiedliche Workflowsprachen in den meisten Fällen unterschiedliche Ausführungsumgebungen, also spezielle Workflow Engines, benötigen. Durch die Verwendung von Workflow Grammatiken kann erreicht werden, dass Prozessmodelle unterschiedlicher Modellierungssprachen auf der gleichen Workflow Engine ausgeführt werden können. Werden Prozessmodelle vor ihrer Ausführung in Workflow Grammatiken transformiert, dann wird zur Ausführung dieser Prozessmodelle lediglich eine Workflow Engine für Workflow Grammatiken benötigt.

Abbildung 8 stellt dar, wie Prozessmodelle unterschiedlicher Modellierungssprachen auf einer einzigen Workflow Engine ausgeführt werden können. Eine ähnliche Motivation lag dem Entwurf von BPEL zugrunde. Im Bereich der Business Workflows existieren vielfältige Modellierungssprachen. Zur Ausführung der mit unterschiedlichen Sprachen erstellten Prozessmodelle werden unterschiedliche Workflow Engines benötigt. BPEL wurde mit dem Ziel entworfen, eine einheitliche

Modellierungssprache für Business Workflows zu bieten. Es ist möglich, die Prozessmodelle unterschiedlicher Modellierungssprachen auf BPEL Prozessmodelle abzubilden und die so auf einer einzigen Workflow Engine auszuführen. Der Unterschied zu Workflow Grammatiken besteht darin, dass BPEL mit Blick auf Business Workflows entwickelt wurde. Die entsprechenden Modellierungssprachen sind meist imperativ und kontrollflussgetrieben, das gilt auch für BPEL. Der Entwurf von Workflow Grammatiken erweitert den Fokus dagegen auf möglichst alle Arten von Prozessbeschreibungssprachen, also beispielsweise auch datengetriebene oder deklarative Sprachen. Sie alle sollen auf Workflow Grammatiken abbildbar sein.

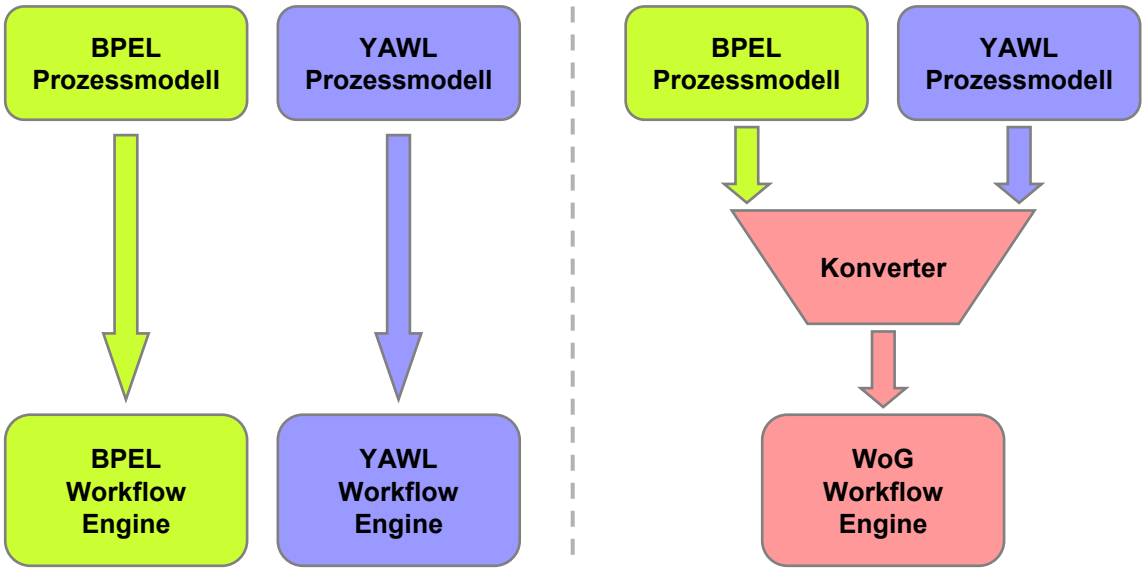


Abbildung 8: Einheitliche Ausführung von Prozessmodellen

Workflow Grammatiken beschreiben Prozesse sehr feingranular. Dadurch wird erreicht, dass auch Prozessmodelle unterschiedlichster Sprachen als Workflow Grammatiken dargestellt werden können. Die feine Granularität von Workflow Grammatiken äußert sich unter anderem darin, dass jede Produktionsregel einer solchen Grammatik einen einzelnen Schritt der Prozessausführung beschreibt. Workflow Grammatiken sind damit teilweise vergleichbar mit Bytecode, einem Zwischencode zwischen Maschinensprache und einer höheren Programmiersprache [HMG11]. Dieser besteht aus wenigen einfachen Befehlen. Einzelne Befehle einer höheren Programmiersprache werden durch mehrere Befehle in Bytecode beschrieben.

3.2. Grundlagen

In dieser Arbeit und insbesondere in diesem Kapitel werden sowohl Variablen als auch Aktivitäten sehr vereinfacht dargestellt. Workflow Grammatiken verwenden als Datenmodell XML Schema, Ausdrücke werden dazu passend als XPath Ausdrücke formuliert. Die Kommunikation erfolgt über Web Services. Workflow Grammatiken übernehmen dabei die entsprechenden Konzepte von BPEL. Die konkreten Details dieser Aspekte spielen im Folgenden aber keine wesentliche Rolle und werden hier nicht näher betrachtet.

Workflow Grammatiken bestehen, wie jede formale Grammatik auch, aus Terminalen, Nichtterminalen und den Produktionsregeln. Das Erzeugen eines Wortes mit den Regeln einer Workflow Grammatik repräsentiert die Ausführung des durch diese Grammatik beschriebenen Prozesses. Die Ableitung eines Wortes, also eine Abfolge von Teilworten und der Anwendung von Regeln auf diese Teilworte, beschreibt die Ausführung eines Prozesses. Jede angewendete Regel repräsentiert eine Aktion, die

während der Prozessausführung durchgeführt wurde. Jedes Teilwort beschreibt einen konkreten Prozesszustand. Die in einem Teilwort enthaltenen Nichtterminale könne unterschiedliche Bedeutungen haben, Terminale repräsentieren immer beendete Aktivitäten. Ein von einer Workflow Grammatik erzeugtes Wort repräsentiert eine beendete Prozessinstanz, es enthält alle in dieser Instanz ausgeführten Aktivitäten.

Der Zusammenhang zwischen der Ausführung eines Prozesses und der Anwendung der Regeln einer Workflow Grammatik wird zu Beginn von Kapitel 4 detailliert dargestellt. Ein Teilwort repräsentiert einen Prozesszustand. Die Anwendung einer Regel beschreibt einen Schritt in der Ausführung eines Prozessmodells.

3.3. Typen von Nichtterminalen

Das zentrale Modellierungskonstrukt in Workflow Grammatiken sind die Produktionsregeln. Bevor diese beschrieben werden, sollen zunächst die unterschiedlichen Typen von Nichtterminale kurz vorgestellt werden. Tabelle 1 gibt eine kurze Übersicht über die verschiedenen Nichtterminaltypen von Workflow Grammatiken.

	Symbol	Name	Beschreibung
Aktivitäten	C_i, c_i	Call	Aufruf eines Web Services
	I_i, i_i	In	Empfang einer Nachricht
	O_i, o_i	Out	Senden einer Antwortnachricht
	A_i, a_i	Assign	Zuweisung
	E_i, e_i	Evaluation	Auswertung einer booleschen Bedingung
	W_i, w_i	Wait	Warten
	Q_i, q_i	Quit	Prozess vorzeitig beenden
	S	Startsymbol	Startsymbol
	V_i	Variable	Variable
	P_i, P_{im}	Placeholder	Platzhalter, mobiler Platzhalter
	S_{il}, S_{ir}	Scope	Lebensbereich für Variablen
	T, F	True, False	Wahr, Falsch
	F_i	Fault	Fehlertyp
	M_i	MessageInterface	Nachrichtenempfang
	R_i	Result	Ergebnisvariable
	L	Life	Normale Prozessausführung
	K_l, K, K_r	Kill	Beenden des Prozesses

Tabelle 1: Nichtterminaltypen von Workflow Grammatiken

Die erste Gruppe der Nichtterminale repräsentiert Aktivitäten. Aktivitäts- Nichtterminale werden mit Großbuchstaben dargestellt, sie repräsentieren aktivierte Aktivitäten. Zu jedem Aktivitäts-Nichtterminal gibt es ein entsprechendes Terminal. Terminale werden mit Kleinbuchstaben dargestellt und repräsentieren beendete Aktivitäten.

Eine *Call* Aktivität ruft einen Web Service auf. Eine *In* Aktivität empfängt eine Nachricht über eine Web Service Schnittstelle. Eine *Out* Aktivität sendet eine Antwortnachricht, diese gehört immer zu einer zuvor über eine *In* Aktivität empfangenen Nachricht.

Eine *Assign* Aktivität manipuliert Variablen. Daten können gelesen und geschrieben werden. Eine Aktivität vom Typ *Evaluation* wertet einen booleschen Ausdruck aus. Eine *Wait* Aktivität wartet. Es können entweder ein Zeitraum oder ein Zeitpunkt angegeben werden. Eine Aktivität vom Typ *Quit* beendet den Prozess vorzeitig.

Platzhalter sind Nichtterminal ohne weitere Semantik. *Mobile Platzhalter* können durch entsprechende Regeln innerhalb eines Teilwortes frei bewegt werden. Ein *Scope* repräsentiert den Lebensbereich von Variablen und wird durch eine linke und eine rechte Grenze dargestellt. Die Nichtterminale *T* und *F* beschreiben die booleschen Werte „true“ (wahr) und „false“ (falsch). Nichtterminale vom Typ *Fault* signalisieren Fehler eines bestimmten Typs. Nichtterminale vom Typ *MessageInterface* repräsentieren den Empfang einer Nachricht über eine bestimmte Schnittstelle. Ein *Result* Nichtterminal beschreibt eine bestimmte Art von Variablen, deren Inhalt für die Ausführung einer Workflow Grammatik von Bedeutung ist und deren Wert von einer entsprechenden Workflow Engine gelesen und interpretiert werden muss. Das Nichtterminal vom Typ *Lebensmarker* markiert die Regeln, welche die normale Prozessausführung beschreiben. Dies steht im Gegensatz zum Nichtterminal vom Typ *Killmarker* sowie den dazugehörigen linken und rechten Grenzen. Diese realisieren das Beenden eines laufenden Prozesses.

3.4. Produktionsregeln

Die Modellierung von Prozessmodellen erfolgt in Workflow Grammatiken durch die Produktionsregeln. Im Folgenden wird beschrieben, wie grundlegende Modellierungskonstrukte in Workflow Grammatiken dargestellt werden. Zunächst wird jedoch noch eine wichtige Eigenschaft beim Aufbau von Produktionsregeln für Workflow Grammatiken vorgestellt.

In einer Produktionsregel wird im Allgemeinen immer nur eine einzelne Aktion modelliert. Dies folgt aus der allgemeinen Semantik von Grammatiken. Die Anwendung einer Produktionsregel entspricht einer atomaren Aktion. Das Ersetzen der linken durch die rechte Seite geschieht in einem Schritt. Innerhalb einer einzigen Produktionsregel können damit nicht mehrere Aktionen modelliert werden, bei denen die Reihenfolge der Ausführung wichtig ist.

Die Produktionsregeln einer Workflow Grammatik, mit denen die Struktur eines Prozesses modelliert wird, werden als *Prozesstrukturregeln* bezeichnet. Neben diesen Regeln gibt es noch die sogenannten *generischen Regeln*. Sie modellieren nicht die eigentliche Struktur des Prozesses sondern zusätzliche allgemeingültige Aspekte. Ein Beispiel dafür ist die freie Verfügbarkeit von Variablen. Nichtterminale, welche Variablen repräsentieren, müssen innerhalb eines Teilwortes frei beweglich sein. Nur dadurch ist es möglich, den Zugriff unterschiedlicher Aktivitäten auf die gleiche Variable zu modellieren. Dieses Verhalten von Variablen Nichtterminale wird durch die in Regelmenge 1 gezeigten generische Regeln definiert. Generische Regeln werden in einer Workflow Grammatik nicht explizit als Teil der Produktionsregeln angegeben. Sie werden in [Vuk11] allgemein definiert und beschrieben.

$V_i\ X \rightarrow X\ V_i$
$X\ V_i \rightarrow V_i\ X$
Für alle Variablen V_i und alle anderen Terminale oder Nichtterminale X.

Regelmenge 1: Generische Regeln für Variablen

Aktivierung und Beendigung von Aktivitäten

Aktivierte Aktivitäten werden durch Aktivitäts- Nichtterminale repräsentiert. Die Aktivierung einer Aktivität wird durch eine Regel modelliert, in der das entsprechende Nichtterminal erzeugt wird. Es kommt auch der rechten Seite der Regel vor, auf der linken Seite dagegen nicht. Die Beendigung

einer Aktivität wird durch eine Regel beschrieben, welche ein Aktivitäts- Nichtterminal durch das entsprechende Terminal ersetzt. Beispielregeln für das Aktivieren und Beenden der Aktivität A_1 sind in Regelmenge 2 dargestellt.

Aktivieren der Aktivität A_1:	…	→	… A_1 …
Beenden der Aktivität A_1:	… A_1 …	→	… a_1 …

Regelmenge 2: Aktivieren und Beenden einer Aktivität

Sequenz

Die Hintereinanderausführung von Aktivitäten wird durch die Verwendung von Platzhaltern, also Nichtterminalen ohne weitere Semantik, realisiert. Sie verknüpfen die Beendigungsregel einer Aktivität mit der Aktivierungsregel ihrer Folgeaktivität. Regelmenge 3 zeigt als Beispiel die Regeln für die Hintereinanderausführung der Aktivitäten A_1 und A_2. In der Beendigungsregel von A_1 wird der Platzhalter P_{A2} erzeugt. Dieser wird in der Aktivierungsregel von A_2 benötigt. Die Aktivität A_2 kann damit erst dann aktiviert werden, wenn die Aktivität A_1 beendet wurde.

Beenden der Aktivität A_1:	… A_1 …	→	… a_1 P_{A2} …
Aktivieren der Aktivität A_2:	… P_{A2} …	→	… A_2 …

Regelmenge 3: Hintereinanderausführung zweier Aktivitäten

Zu Beginn von Kapitel 3.4 wurde beschrieben, dass Aktionen, zwischen denen ein zeitlicher Zusammenhang besteht, in Workflow Grammatiken nicht in einer einzelnen Regel modelliert werden dürfen. Regelmenge 4 demonstriert dieses Prinzip hier noch einmal an einem kleinen Beispiel. Es wird angenommen, dass die Aktivität A_2 nach der Aktivität A_1 ausgeführt werden soll. Erst wenn A_1 beendet wurde darf A_2 aktiviert werden. Die erste dargestellte Regel modelliert dieses nicht korrekt. Sie beschreibt die Beendigung von A_1 und die Aktivierung von A_2 in einem einzigen Schritt. Die beiden folgenden Regeln modellieren den beschriebenen Prozessausschnitt dagegen korrekt. Die erste dieser beiden Regeln beschreibt die Beendigung der Aktivität A_1 und die Erzeugung des Platzhalters P_{A2}. Erst nach der Abarbeitung dieser Regel kann im nächsten Schritt mit der letzten dargestellten Regel die Aktivität A_2 aktiviert werden. Damit ist der zeitliche Zusammenhang zwischen der Beendigung von A_1 und der Aktivierung von A_2 korrekt dargestellt.

Falsch:	… A_1 …	→	… a_1 A_2 …
Richtig:	… A_1 …	→	… a_1 P_{A2} …
	… P_{A2} …	→	… A_2 …

Regelmenge 4: Workflow Grammatik Regelaufbau

Datenfluss

Datenfluss wird in Workflow Grammatiken durch den Zugriff auf Variablen modelliert. Dazu werden Variablen in beide Seiten einer Aktivierungs- oder Beendigungsregel eingefügt. Ein Beispiel dafür wird in Regelmenge 5 gezeigt. Die Aktivität A_1 liest aus Variable V_1 und schreibt in Variable V_2. Lesender Zugriff wird immer in der Aktivierungsregel dargestellt, schreibender Zugriff in der Beendigungsregel.

Lesender Zugriff auf V_1:	… V_1 …	→	… V_1 A_1 …
Schreibender Zugriff auf V_2:	… V_2 A_1 …	→	… V_2 a_1 …

Regelmenge 5: Lesender und schreibender Datenzugriff

Verzweigung und Zusammenführung

Eine Verzweigung des Kontrollflusses in mehrere parallele Stränge wird in Workflow Grammatiken durch Regeln dargestellt, die auf ihrer rechten Seite die Platzhalter für mehr als eine Folgeaktivität erzeugen. Die anschließende Zusammenführung paralleler Stränge wird mit Hilfe weiterer Platzhalter modelliert. Jeder der parallelen Stränge erzeugt bei der Beendigung seiner letzten Aktivität einen Platzhalter. Die Aktivierungsregel derjenigen Aktivität, an der die Zusammenführung stattfindet, benötigt auf ihrer linken Seite die Platzhalter aller parallelen Stränge.

Regelmenge 6 zeigt die Modellierung von Verzweigung und Zusammenführung an einem einfachen Beispiel. Nach der Beendigung der Aktivität A_1 verzweigt sich der Kontrollfluss in zwei parallele Zweige. Im ersten Zweig wird die Aktivität A_2 aktiviert, im zweiten Zweig die Aktivität A_3. Beide Aktivitäten erzeugen bei ihrem Beenden einen Platzhalter. Die Zusammenführung der parallelen Zweige erfolgt an der Aktivität A_4. Sie benötigt für ihre Aktivierung die Platzhalter beider Zweige, P_{A4m} und P_{A4}.

Verzweigung:	… A_1 …	→	… a_1 P_{A2} P_{A3} …
Zweig 1:	… P_{A2} …	→	… A_2 …
	… A_2 …	→	… a_2 P_{A4m} …
Zweig 2:	… P_{A3} …	→	… A_3 …
	… A_3 …	→	… a_3 P_{A4} …
Zusammenführung:	… P_{A4m} P_{A4} …	→	… A_4 …

Regelmenge 6: Verzweigung und Zusammenführung

Die für eine Zusammenführung erzeugten Platzhalter sind sogenannte mobile Platzhalter. Dies wird durch ein „m“ im Index des entsprechenden Platzhalters gekennzeichnet. Mobile Platzhalter können durch entsprechende generische Regeln im gesamten Teilwort frei bewegt werden. Im Beispiel ist der Platzhalter P_{A4m} ein solcher mobiler Platzhalter. Genau einer der für die Zusammenführung erzeugten Platzhalter ist kein mobiler Platzhalter. Er kennzeichnet den Punkt innerhalb eines Teilwortes, an dem die Zusammenführungsregel angewendet wird.

Bedingter Kontrollfluss

Die Modellierung von bedingtem Kontrollfluss kann in Workflow Grammatiken auf verschiedenen Arten erfolgen. Kontrollfluss kann abhängig sein von Booleschen Bedingungen, vom Auftreten von Fehlern unterschiedlicher Typen oder vom Empfang einer Nachricht über eine bestimmte Schnittstelle. Der Aufbau der entsprechenden Regeln folgt dem jeweils gleichen Prinzip, unterscheidet sich jedoch ein einigen Punkten.

Kontrollfluss abhängig von booleschen Bedingungen wird durch eine Evaluation Aktivität, eine Ergebnisvariable und die Nichtterminal T und F dargestellt. Ein Beispiel ist in Regelmenge 7 dargestellt. Die Aktivität E_1 wertet eine boolesche Bedingung aus. Sie erzeugt beim Beenden die Ergebnisvariable R_1 und legt das Ergebnis der Auswertung in dieser ab. Anschließend können zwei unterschiedliche Regeln angewendet werden. Das Nichtterminal T identifiziert diejenige Regel, die angewendet wird, wenn das in R_1 abgelegte Ergebnis der Auswertung „wahr („true“) ist. Das Nichtterminal F identifiziert die Regel für den Fall, dass die Auswertung „falsch“ („false“) ergeben hat. Welche der beiden Regeln angewendet wird, kann erst zur Laufzeit entschieden werden.

Auswertung der Bedingung:	… E_1 …	→	… e_1 R_1 …
Ergebnis „wahr“:	… R_1 …	→	… T P_1 …
Ergebnis „falsch“:	… R_1 …	→	… F P_2 …

Regelmenge 7: Bedingter Kontrollfluss, Boolesche Bedingung

Das Auftreten von Fehlern wird in Workflow Grammatiken durch das Erzeugen eines Nichtterminals modelliert, welches den Typ des entsprechenden Fehlers repräsentiert. Kann eine Aktivität einen Fehler erzeugen, so wird dies durch eine Beendigungsregel beschrieben, die ein Fehler- Nichtterminal erzeugt. Eine Beendigungsregel ohne Fehler- Nichtterminal beschreibt das normale Beenden einer Aktivität. Regelmenge 8 zeigt mehrere Beendigungsregeln für die Aktivität A_1. Die erste Regel beschreibt das normale Beenden von A_1. Sie wird nur dann angewendet, wenn die Aktivität A_1 keinen Fehler produziert. Die weiteren Regeln werden genau dann angewendet, wenn die Aktivität A_1 beim Beenden einen Fehler vom Typ F_1 oder F_2 erzeugt.

Beendigung ohne Fehler:	… A_1 …	→	… a_1 P_1 …
Beendigung mit Fehler F_1:	… A_1 …	→	… a_1 F_1 P_2 …
Beendigung mit Fehler F_2:	… A_1 …	→	… a_1 F_2 P_3 …

Regelmenge 8: Bedingter Kontrollfluss, Fehlertypen

Das Empfangen von Nachrichten wird in Workflow Grammatiken mit Aktivitäten vom Typ „In“ modelliert. Eine solche Aktivität kann mehrere Schnittstellen definieren, über welche Nachrichten empfangen werden können. Die Aktivität wird beendet, sobald eine Nachricht empfangen wurde. Die empfangene Nachricht sowie die Informationen über die Schnittstelle, über welche sie empfangen wurde, werden zunächst in einer Ergebnisvariablen zwischengespeichert. Abhängig von der Schnittstelle, über welche die Nachricht empfangen wurde, können anschließend unterschiedliche Regeln angewendet werden. Ein Beispiel dafür ist in Regelmenge 9gezeigt. Die „In“ Aktivität I_1 kann Nachrichten über zwei unterschiedliche Schnittstellen M_1 und M_2 empfangen. Nach dem Beenden von I_1 wird die empfangene Nachricht zunächst in der Ergebnisvariablen R_1 abgelegt. Zusätzlich wird in R_1 hinterlegt, über welche Schnittstelle die Nachricht empfangen wurde. Die Nichtterminale M_1 und M_2 signalisieren, welche der beiden möglichen Regeln nach dem Beenden von I_1 angewendet werden soll.

Nachricht empfangen:	… I_1 …	→	… i_1 R_1 …
Schnittstelle M_1:	… V_1 R_1 …	→	… V_1 M_1 P_1 …
Schnittstelle M_2:	… V_2 R_1 …	→	… V_2 M_2 P_2 …

Regelmenge 9: Bedingter Kontrollfluss, Empfang einer Nachricht

Lebensbereich für Variablen (Scope)

Der Lebensbereich von Variablen wird durch einen sogenannten Scope bestimmt. Variablen werden zusammen mit einem Scope erzeugt und zusammen mit einem Scope wieder gelöscht. In Workflow Grammatiken wird ein Scope durch eine linke und eine rechte Grenze dargestellt. Ein Scope wird genau dann beendet, wenn innerhalb seiner Grenzen alle Aktivitäten abgearbeitet wurden. Regelmenge 10 zeigt ein Beispiel für die Verwendung von Scope und Variablen. In der ersten Regel werden die Grenzen des Scopes S_1 sowie die in ihm lebenden Variablen V_1 und V_2 erzeugt. Innerhalb des Scopes können zusätzlich noch Platzhalter für den weiteren Kontrollfluss innerhalb des Scopes erzeugt werden. Das Beenden eines Scopes wird in Workflow Grammatiken durch generische Regeln ermöglicht. Diese beschreiben, dass eine linke Scopegrenze nach rechts wandern kann, allerdings nur über Terminale. Ist der Kontrollfluss innerhalb des Scopes S_1 beendet, dann können mit der zweiten Regel sowohl der Scope als auch die darin lebenden Variablen V_1 und V_2 gelöscht werden.

Scope und Variablen erzeugen:	…	→	S_{1l} V_1 V_2 … S_{1r}
Scope und Variablen löschen:	S_{1l} V_1 V_2 S_{1r}	→	P_1

Regelmenge 10: Scope und Variablen

Vorzeitiges Beenden einer Prozessinstanz

Die Aktivität „Quit“ wird in Workflow Grammatiken dazu verwendet, eine Prozessinstanz explizit und vorzeitig zu beenden. Das Aktivieren einer solchen Aktivität bewirkt das sofortige Beenden aller weiteren Aktivitäten und stoppt damit den gesamten Prozess. Obwohl dies in den bisherigen Beispielen nicht dargestellt wurde, besitzen alle Produktionsregeln, welche den normalen Kontrollfluss beschreiben, auf beiden Seiten ein spezielles Nichtterminal, den sogenannten Lebensmarker. Das Aktivieren einer „Quit“ Aktivität löscht diesen Lebensmarker. Dadurch wird verhindert, dass der restliche Prozess weiter ausgeführt wird. Gleichzeitig wird ein Killmarker erzeugt, ein spezielles Nichtterminal, welches zusammen mit entsprechenden generischen Regeln das Beenden des aktuellen Prozesses modelliert.

Normaler Kontrollfluss:	… L A_1 …	→	… L a_1 P_{Q1} …
Prozessausführung stoppen:	… L P_{Q1} …	→	… K Q_1 …
Prozessende:	… K_1 K K_r …	→	q_1

Regelmenge 11: Prozessinstanz beenden

3.5. Zusammenfassung

Der Entwurf von Workflow Grammatiken wurde damit motiviert, eine Basis zu schaffen, auf die möglichst viele Workflowsprachen abgebildet werden können. Damit soll zum einen die Vergleichbarkeit und sinnvolle Klassifizierung unterschiedlichster Workflowsprachen erreicht werden. Zum anderen wird es durch Workflow Grammatiken möglich, Prozessmodelle unterschiedlicher Modellierungssprachen auf nur einer Workflow Engine auszuführen. Der Aspekt der Ausführung von Workflow Grammatiken steht im Zentrum dieser Arbeit.

Nach einer kurzen Beschreibung der Entwurfsprinzipien von Workflow Grammatik wurden die in diesen Grammatiken verwendeten Typen von Nichtterminalen vorgestellt. Ein Teil dieser Nichtterminale repräsentiert unterschiedliche Aktivitäten wie beispielsweise den Aufruf eines Web Services oder eine Zuweisung. Die Nichtterminale werden verwendet, um Produktionsregeln zu formulieren. Diese Regeln beschreiben die Struktur eines Prozesses. Die Modellierungskonstrukte von Workflow Grammatiken, also verschiedenen Arten von Regeln, wurden kurz vorgestellt und an jeweils einem kleinen Beispiel beschrieben.

4 Anwendung von Grammatiken zur Erzeugung von Wörtern

Die in Kapitel 2 vorgestellten Automatenmodelle akzeptieren die jeweilige Sprachtypen. Ihre Anwendung besteht darin, zu einem gegebenen Wort in endlicher Zeit zu entscheiden, ob dieses Wort zu einer Sprache gehört oder nicht. Derartige Automaten bezeichnet man, wie bereites erwähnt, folgegemäß auch als Akzeptoren.

In den folgenden Unterkapiteln wird das algorithmische Erzeugen von Wörtern mit Hilfe von Grammatiken näher untersucht. Zuvor werden diese Betrachtungen aber zunächst motiviert und mit Workflow Grammatiken in Verbindung gesetzt. Zusätzlich werden grundlegende Forderungen identifiziert und formuliert, welche bei der Anwendung von Worterzeugungsmechanismen im Kontext von Workflow Grammatiken zu beachten sind.

Akzeptoren und Generatoren

Im Kontext von Workflow Grammatiken soll eine Grammatik verwendet werden, um ein Wort zu erzeugen. Die in Kapitel 2.2 vorgestellten Automatenmodelle arbeiten ebenfalls auf Grammatiken. Sie erzeugen dabei aber keine Wörter, sondern sie erkennen oder akzeptieren Wörter. Der prinzipielle Unterschied zwischen akzeptierenden und erzeugenden Verfahren ist in Abbildung 9 schematisch dargestellt.

Abbildung 9: Akzeptor und Generator

Zu einer gegebenen Grammatik kann immer ein entsprechender Akzeptor konstruiert werden. Dieser Akzeptor bekommt ein Wort übergeben und akzeptiert es genau dann, wenn es mit der dazugehörigen Grammatik produziert werden kann. Der Navigator einer durch eine Workflow Grammatik definierten Prozessinstanz arbeitet dagegen als Generator. Er verwendet eine gegebene Grammatik, um ein Wort zu produzieren.

Der in Abbildung 9 auf der rechten Seite dargestellte allgemeine Generator kann alle Wörter der durch die Grammatik beschriebenen Sprache erzeugen. Welches Wort genau er erzeugt, ist im allgemeinen Fall nichtdeterministisch. In jedem Schritt einer Ableitung kann es möglich sein, auf das aktuelle Teilwort mehrere alternative Produktionsregeln anzuwenden. Die Entscheidung, welche der Regeln im nächsten Schritt angewendet wird, kann bei einer allgemeinen Grammatik nur nichtdeterministisch getroffen werden.

Der Unterschied zwischen der Erzeugung eines Wortes einer allgemeinen Grammatik und der Erzeugung eines Wortes einer Workflow Grammatik liegt in der besonderen Semantik von Workflow

Grammatiken begründet. Der Aufbau einer Workflow Grammatik ist zudem nicht beliebig, er folgt den in Kapitel 3 vorgestellten Prinzipien. Die Besonderheiten beim Erzeugen von Wörtern mit Workflow Grammatiken werden im folgenden Abschnitt näher betrachtet.

Workflow Grammatiken

In Kapitel 3 wurde das Konzept von Workflow Grammatiken vorgestellt. Workflow Grammatiken beschreiben Prozessmodelle mit Hilfe von formalen Grammatiken. Ein Prozessmodell auszuführen bedeutet, die Regeln einer Grammatik anzuwenden, um damit ein Wort zu produzieren. Ein von einer Workflow Grammatik erzeugtes Wort repräsentiert dann eine abgeschlossene Prozessinstanz. Es beschreibt, welche Aktivitäten während der Ausführung einer Instanz aktiv waren. Ein Teilwort einer Workflow Grammatik beschreibt dagegen eine laufende Prozessinstanz. Nichtterminale innerhalb dieses Teilwortes können dann als „Punkte“ im Prozess interpretiert werden, an denen die Prozessausführung fortgesetzt werden kann.

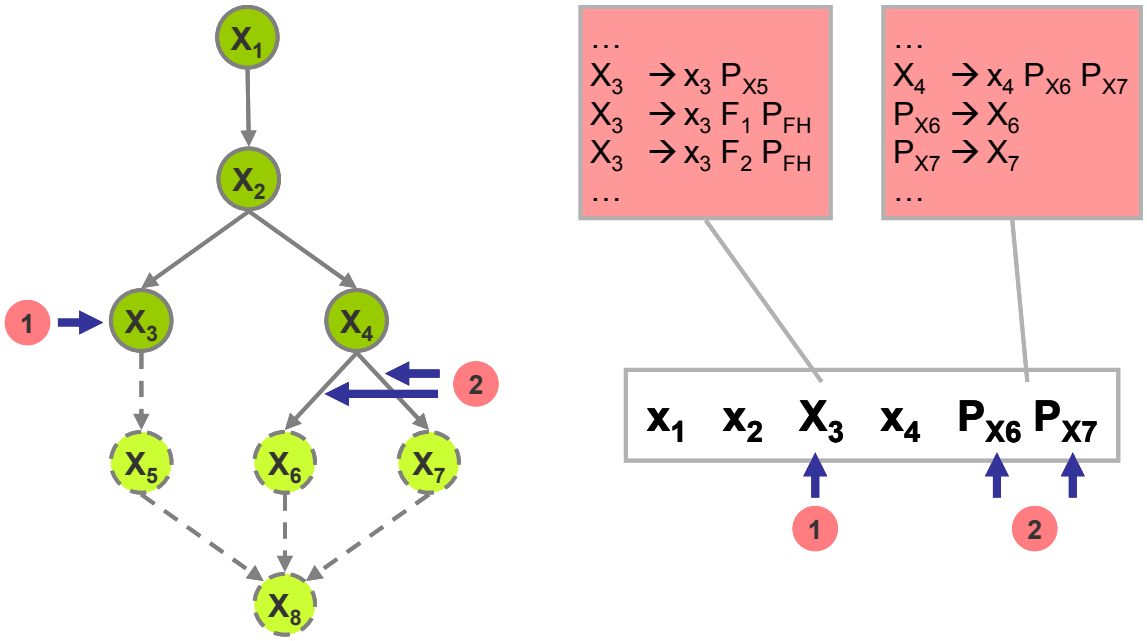


Abbildung 10: Repräsentationen von Prozessinstanzen

Abbildung 10 verdeutlicht den Zusammenhang zwischen einer aktiven Prozessinstanz und dem entsprechenden Teilwort an einem kleinen Beispiel. Auf der linken Seite ist eine Prozessinstanz in Form eines Graphen dargestellt. Die gestrichelten Aktivitäten wurden noch nicht aktiviert, alle anderen Aktivitäten sind aktiv oder bereits beendet. Die nummerierten Pfeile beschreiben den aktuellen Ausführungszustand. Pfeil (1) zeigt, dass die Aktivität X_3 gerade aktiv ist. Pfeil (2) zeigt, dass die Aktivität X_4 beendet wurde, aber die Nachfolgeaktivitäten X_6 und X_7 noch nicht aktiviert wurden. Auf der rechten Seite ist das entsprechende Teilwort dargestellt. Die Terminale beschreiben die bereits beendeten Aktivitäten, das Nichtterminal X_3 stellt die laufende Aktivität X_3 dar. Die Platzhalter P_{X6} und P_{X7} stellen den aktiven Kontrollfluss zwischen der Aktivität X_4 den Folgeaktivitäten X_6 und X_7 dar. Sie werden bei der Beendigung der Aktivität X_4 erzeugt und ermöglichen damit die anschließende Aktivierung der Aktivitäten X_6 und X_7. Die entsprechenden Regeln sind ebenfalls mit dargestellt.

Anhand des gezeigten Beispiels lassen sich einige grundlegende Aussagen über die Ausführung von Workflow Grammatiken formulieren.

1. *Regeln bzw. Nichtterminale können blockiert sein*
   Nichtterminale können laufende Aktivitäten repräsentieren (die Gruppe solcher Nichtterminale wird

in Workflow Grammatiken auch *Aktivitäts- Nichtterminale* genannt). Das bedeutet, dass diese Nichtterminale in einem Teilwort so lange, wie die entsprechenden Aktivitäten aktiv sind, nicht aus dem Teilwort entfernt werden dürfen. Alle Regeln der Grammatik, die dies bewirken würden, müssen für den betreffenden Zeitraum blockiert werden.

2. *Freie Auswahl alternativer Regeln*
   Ein Teilwort kann zu einem bestimmten Zeitpunkt mehrere Nichtterminale enthalten. Das bedeutet, dass mehrere alternative Regelanwendungen möglich sind. Bezogen auf die durch das Teilwort repräsentierte Prozessinstanz bedeutet dies, dass zu einem bestimmten Zeitpunkt mehrere Aktionen möglich sein können.
   In dem in Abbildung 10 dargestellten Beispiel enthält das Teilwort auf der rechten Seite die beiden Platzhalter P_{X6} und P_{X7}. Auf jeden Platzhalter kann eine Regel angewendet werden. Im dargestellten Prozessmodell entspricht diese Situation der Verzweigung des Kontrollflusses in eine Parallelität. Welche der beiden Aktivitäten X_6 und X_7 dabei zuerst aktiviert wird, ist durch das Prozessmodell nicht vorgegeben. Dies bedeutet für das Teilwort, dass frei entscheidbar ist, ob im nächsten Schritt eine Regel auf den Platzhalter P_{X6} oder auf den Platzhalter P_{X7} angewendet wird.

3. *Bedingte Auswahl alternativer Regeln mit gleicher linker Seite*
   Im in Abbildung 10 dargestellten Beispiel ist die Aktivität X_3 noch aktiv. Die entsprechenden Regeln sind also blockiert, sie dürfen nicht angewendet werden. Sobald die Aktivität beendet ist, dürfen Regeln auf das Nichtterminal X_3 angewendet werden. Im dargestellten Beispiel gibt es mehrere alternative Regeln, die in diesem Fall angewendet werden können.
   In Workflow Grammatiken sind alternative Regeln mit der gleichen linken Seite immer so aufgebaut, dass die Auswahl von genau einer Regel zur Laufzeit immer deterministisch möglich ist. Im dargestellten Beispiel existieren mehrere Beendigungsregeln für die Aktivität bzw. das Nichtterminal X_3. Es gibt dabei genau eine Beendigungsregel ohne Vorkommen eines Fehler Nichtterminals sowie mehrere Beendigungsregel mit paarweise unterschiedlichen Fehler Nichtterminalen. Wird die Aktivität X_3 zur Laufzeit der Prozessinstanz mit einem Fehler beendet, dann wird die Beendigungsregel mit dem passenden Fehler Nichtterminal gewählt. Wird sie ohne Fehler beendet, wird die entsprechende Beendigungsregel ohne Fehler Nichtterminal gewählt.
   Alternative Regeln dieser Art können in Workflow Grammatiken, wie in Kapitel 3.4 beschrieben, neben auftretenden Fehlern auch abhängig sein von der Auswertung von Ausdrücken bzw. booleschen Werten oder vom Empfangen einer Nachricht.

Zunächst ist festzuhalten, dass die Entscheidung, welche Produktionsregel im nächsten Schritt auf ein Teilwort angewendet wird, in zwei Schritten abläuft. Jeder dieser Schritt hat, bezogen auf die Prozessausführung, eine unterschiedliche Bedeutung.

Im ersten Schritt muss entschieden werden, an welcher Stelle des Teilwortes eine Produktionsregel angewendet werden soll. Dies entspricht bei Workflow Grammatiken der Entscheidung, welche von mehreren möglichen Aktionen der Navigator in der Prozessinstanz als nächstes ausführen soll. Dabei müssen insgesamt alle Aktionen ausgeführt werden, es wird lediglich eine Entscheidung über ihre Reihenfolge getroffen.

Im zweiten Schritt muss entschieden werden, welche aller möglichen Regeln auf eine jetzt fest gewählte Stelle im Teilwort angewendet werden soll. Dies ist immer die Auswahl einer Regel aus einer Menge von Regeln mit der gleichen linken Seite. Bei Workflow Grammatiken kann eine solche Entscheidung zur Laufzeit einer Prozessinstanz immer deterministisch getroffen werden. Sie wird, allgemein formuliert, vom Kontext der aktuell ausgeführten Prozessinstanz bestimmt.

Aus den soeben hergeleiteten Aussagen über die Erzeugung von Wörtern im Kontext von Workflow Grammatiken lassen sich zwei Forderungen formulieren. Ein Mechanismus bzw. Algorithmus zur Erzeugung von Wörtern aus einer Workflow Grammatik muss diese Forderungen erfüllen. Nur in diesem Fall entspricht die Erzeugung eines Wortes aus einer Workflow Grammatik der korrekten Ausführung des durch die Grammatik beschriebenen Prozessmodells.

1. *Wann immer es möglich ist, auf ein Teilwort mindestens eine Regel anzuwenden, dann soll auch eine Regel angewendet werden.*
   Wenn es in einem Teilwort möglich ist, mindestens eine Regel anzuwenden, dann bedeutet dies in Workflow Grammatiken, dass die Ausführung der betreffenden Prozessinstanz fortgesetzt werden kann. Es gibt mindestens einen Punkt im Prozess, an dem der Navigator seine Arbeit fortsetzen kann. Wann immer möglich, soll der Navigator dies auch tun.
   Die Forderung bedeutet insbesondere, dass die Existenz blockierter Nichtterminale in einem Teilwort nicht bewirken darf, dass Regeln, die diese blockierten Nichtterminale nicht beinhalten, ebenfalls blockiert werden. Dieser Fall kann beispielsweise beim Erzeugen von Wörtern aus kontextfreien Grammatiken mit Hilfe eines Kellerautomaten auftreten. Dies wird in Kapitel 4.2 näher beschrieben.
2. *Ist es möglich, Regeln auf verschiedene Teile eines Teilwortes anzuwenden, dann soll frei entscheidbar sein, an welchem Punkt zuerst eine Regel angewendet wird (optionale Forderung).*
   Wenn es in einem Teilwort möglich ist, Regeln an verschiedenen Punkten dieses Teilwortes anzuwenden, dann bedeutet dies bei Workflow Grammatiken, dass es für den Navigator mehrere, sich gegenseitig nicht ausschließende, alternative Möglichkeiten gibt, die Ausführung der Prozessinstanz fortzusetzen. Eine solche Situation ist im in Abbildung 10 gezeigten Beispiel dargestellt. Sowohl die Aktivität X_6 als auch die Aktivität X_7 können im nächsten Schritt aktiviert werden. Der Navigator muss beide Aktivitäten aktivieren, er kann diese aber in unterschiedlicher Abfolge tun.
   Die Erfüllung dieser optionalen Forderung bewirkt, dass es bei der Ausführung einer Prozessinstanz mehr Freiheiten gibt, die Reihenfolge der einzelnen Schritte zu beeinflussen. Diese Freiheiten können unter Umständen dazu genutzt werden, die Ausführung einer Prozessinstanz zu optimieren. Ein Beispiel dafür ist, dass lang laufende Aktivitäten bevorzugt aktiviert werden. Damit kann die Laufzeit einer Prozessinstanz verringert werden.

Zusammengefasst wird also ein effizientes deterministisches Verfahren gesucht, auf Basis einer gegebenen Workflow Grammatik ein Wort zu produzieren. Es wurden zwei Forderungen formuliert, die aus der speziellen Semantik von Workflow Grammatiken resultieren. Das gesuchte Verfahren muss diese Forderungen erfüllen.

In den folgenden Kapiteln wird für die Grammatiktypen der Chomsky Hierarchie untersucht, wie ein solches Verfahren für den jeweiligen Grammatiktyps aussehen könnte. Ein wesentlicher Teil dieser Betrachtung wird der Vergleich mit den bekannten und bereits in Kapitel 2.2 vorgestellten Automatenmodellen zur Erkennung von Wörtern sein.

4.1. Reguläre Grammatiken (Chomsky Typ 3)

Die Produktionsregeln regulärer Grammatiken ersetzen immer genau ein Nichtterminal (linke Seite der Regel). Es wird entweder durch ein Terminal oder durch genau ein Nichtterminal und ein Terminal ersetzt (rechte Seite der Regel). Dies bedeutet, dass, induktiv vom Startsymbol ausgehend, in jedem Schritt einer Ableitung maximal ein Nichtterminal ersetzt werden kann. Jedes Wort der Ableitung enthält genau ein Nichtterminal sowie beliebige Terminale (ausgenommen das Endwort, es besteht nur aus Terminalen).

Daraus folgt, dass schon vor der eigentlichen Anwendung der Grammatik für jede Regel statisch bestimmt werden kann, welche Regeln im nächsten Schritt angewendet werden können. Dieses Prinzip wird in Abbildung 11 an einer Beispielgrammatik dargestellt.

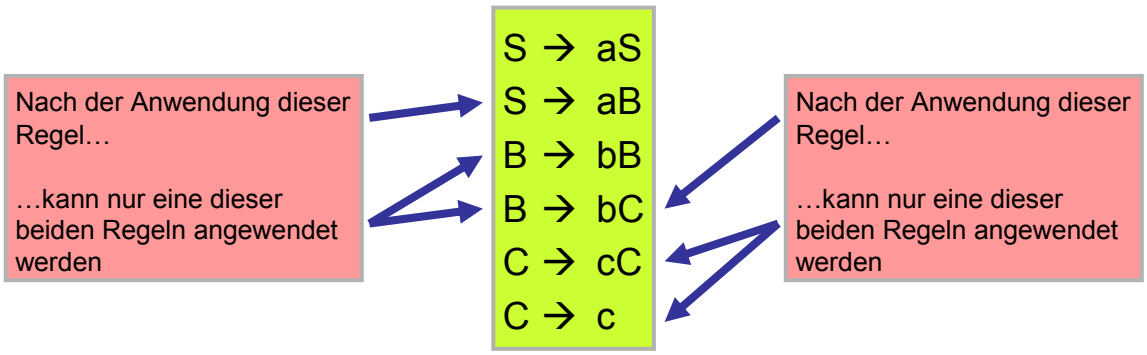


Abbildung 11: Regelanwendung regulärer Grammatiken

Aus dieser Beobachtung lässt sich das Erzeugen von Wörtern einer regulären Grammatik parallel zum Aufbau eines Deterministischen Endlichen Automaten (DEA) bzw. Nichtdeterministischen Endlichen Automaten (NEA) formulieren. Die Konstruktion eines Wörter erzeugenden Automaten entspricht der eines DEA oder NEA, lediglich die Ausführungssemantik unterscheidet sich. Der Automat beginnt im Startzustand. In jedem Zustand wählt er einen der möglichen Übergänge aus, folgt diesem Übergang und schreibt eines der mit dem Übergang assoziierten Zeichen auf das Ausgabeband. Befindet dich der Automat in einem Endzustand, so kann er die Ausführung beenden. Das auf dem Ausgabeband befindliche Wort ist ein gültiges Wort der Sprache. Abbildung 12 vergleicht die Funktionsweise von Akzeptor und Generator für reguläre Sprachen an einem kleinen Beispiel. Der auf der linken Seite dargestellte NEA akzeptiert alle Wörter, die eine Binärzahl ohne führende Nullen darstellen. Der auf der rechten Seite gezeigte Generator kann genau solche Wörter erzeugen.

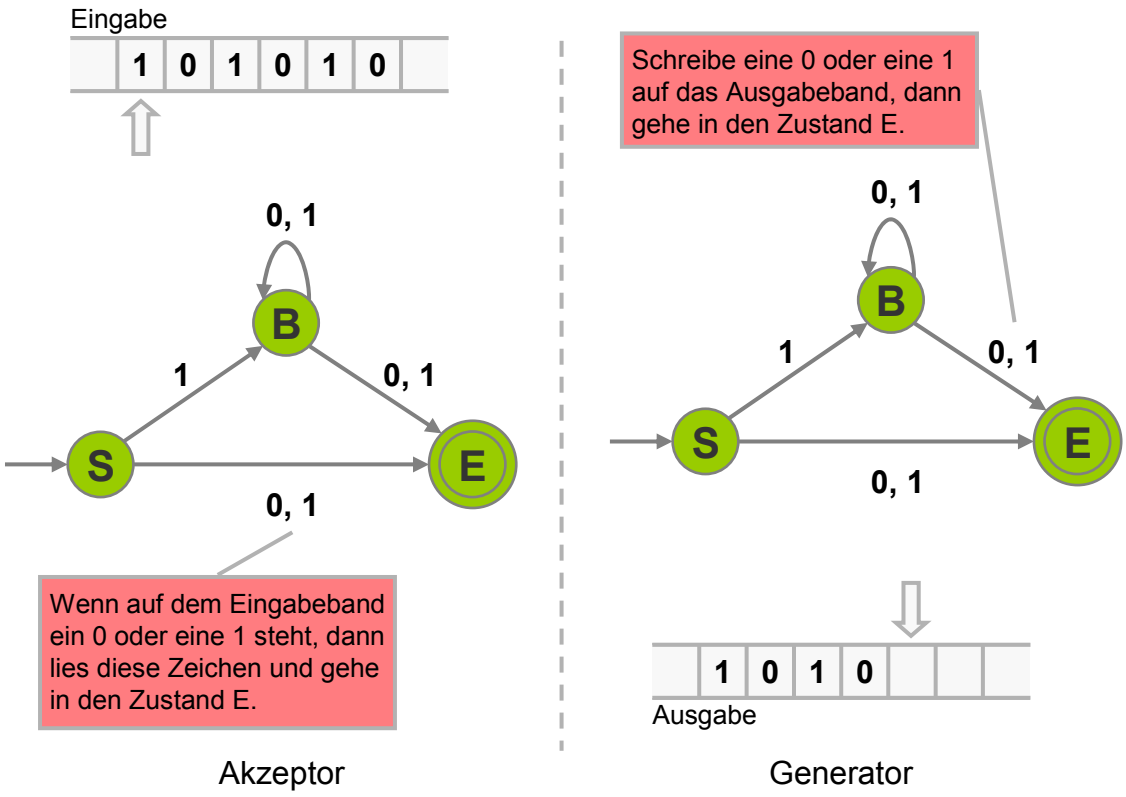


Abbildung 12: Gegenüberstellung Akzeptor und Generator für reguläre Sprachen

Die Konstruktion eines Akzeptors, also eins NEA oder eines DEA, aus einer gegebenen regulären Grammatik ist algorithmisch möglich. Die gleiche Konstruktion kann verwendet werden, um den Zustandsübergangsgraphen für einen Generator zu konstruieren. Der einzige Unterschied besteht, wie bereits erwähnt, in der Interpretation dieses Graphen zur Laufzeit.

Die zu Beginn dieses Kapitels aufgestellten Forderungen an Verfahren zur Erzeugung von Wörtern aus Workflow Grammatiken sind bei dem hier vorgestellten Verfahren für reguläre Grammatiken erfüllt. In jedem Teilwort einer Ableitung eines Wortes aus einer regulären Grammatik gibt es immer genau ein Nichtterminal. Eine Ableitung beginnt immer mit dem Startsymbol. In jedem Schritt wird, ausgehend vom Startsymbol, genau eine Regel angewendet. Jede Regel ersetzt genau ein Nichtterminal durch höchstens ein Nichtterminal, es kann also niemals mehr als ein Nichtterminal in einem Teilwort existieren.

1. *Wann immer es möglich ist, auf ein Teilwort mindestens eine Regel anzuwenden, dann soll auch eine Regel angewendet werden.*
   Es gibt in jedem Teilwort immer nur höchstens ein Nichtterminal. Ist dieses Nichtterminal blockierend, dann blockiert es keine anderen Nichtterminale, denn es gibt keine weiteren Nichtterminale in diesem Teilwort. Ist das Nichtterminal nicht blockierend, dann können auch Regeln auf dieses angewendet werden. Die Forderung ist damit erfüllt.
2. *Ist es möglich, Regeln auf verschiedene Teile eines Teilwortes anzuwenden, dann soll frei entscheidbar sein, an welchem Punkt zuerst eine Regel angewendet wird (optionale Forderung).*
   Es ist niemals möglich, Regeln auf verschiedene Teile eines Teilwortes anzuwenden. Es gibt immer nur höchstens ein Nichtterminal pro Teilwort. Die Forderung ist damit erfüllt.

4.2. Kontextfreie Grammatiken (Chomsky Typ 2)

Die Produktionsregeln Kontextfreier Grammatiken ersetzen immer genau ein Nichtterminal (linke Seite der Regel). Es wird durch eine beliebige, nicht leere Folge von Terminalen und / oder Nichtterminalen ersetzt. Dies bedeutet, dass in jedem Schritt einer Ableitung genau ein Nichtterminal aus dem Teilwort entfernt wird. In jedem Schritt können zusätzlich dazu ein oder mehrere Nichtterminale dem Teilwort hinzugefügt werden. Jedes Wort der Ableitung enthält mindestens ein Nichtterminal (ausgenommen das Endwort, es besteht nur aus Terminalen).

Wörter kontextfreier Sprachen können durch Kellerautomaten erkannt werden (siehe Kapitel 2.2.2.2). Kellerautomaten besitzen, im Unterschied zu den endlichen Automatenmodellen regulärer Sprachen, einen Speicher. Dieser Speicher ist in Form eines Kellers organisiert, es kann lediglich das oberste Zeichen gelesen werden, neue Zeichen können ebenfalls nur oben hinzugefügt werden.

Basierend auf dem Prinzip des Kellerautomaten wird im Folgenden ein Modell eines Generators für kontextfreie Grammatiken entworfen. Ein generierender Kellerautomat besitzt, im Gegensatz zum akzeptierenden Kellerautomat, ein Ausgabe- statt eines Eingabebands. Der Kellerspeicher enthält zu Beginn der Erzeugung eines Wortes das Startsymbol.

In jedem Schritt liest der Automat das oberste Zeichen des Kellers. Handelt es sich um ein Nichtterminal, dann wendet er eine passende Produktionsregel an. Das bedeutet, dass er das Nichtterminal vom Keller entfernt und danach die rechte Seite der angewendeten Regel in umgekehrter Reihenfolge auf den Keller schreibt. Handelt es sich bei dem obersten Zeichen des Kellers dagegen um ein Terminal, dann wird es aus dem Keller entfernt und auf das Ausgabeband geschrieben.

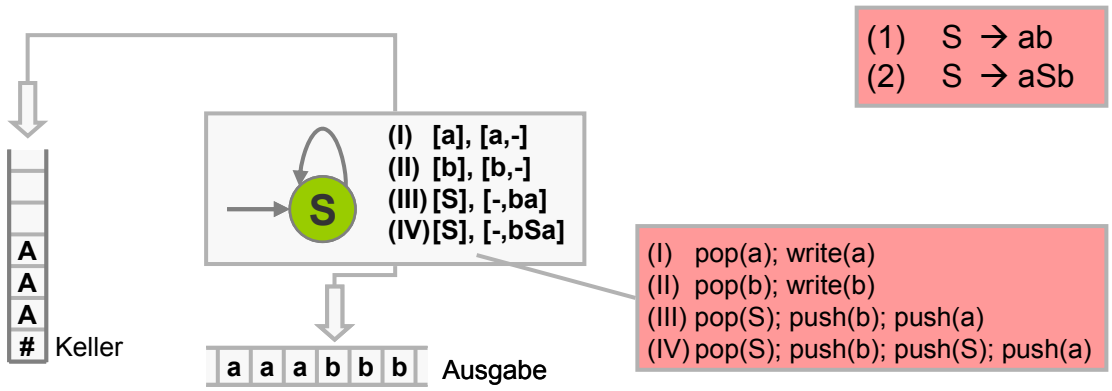


Abbildung 13: Kellerautomat als Generator

In Abbildung 13 ist die Verwendung eines Kellerautomaten als Generator an einem kleinen Beispiel dargestellt. Der Automat greift auf einen Keller lesend und schreibend sowie auf ein Ausgabeband nur schreibend zu. Er besitzt lediglich einen Zustand und vier verschiedenen Übergänge. Die mit den Übergängen verbundenen Kelleroperationen sind rechts vom Automaten dargestellt. Die Operation „pop()“ entfernt das oberste Kellerzeichen, die Operation „push()“ legt ein Zeichen auf den Keller. Der Automat erzeugt genau die Wörter der oben rechts in der Abbildung dargestellten Beispielgrammatik.

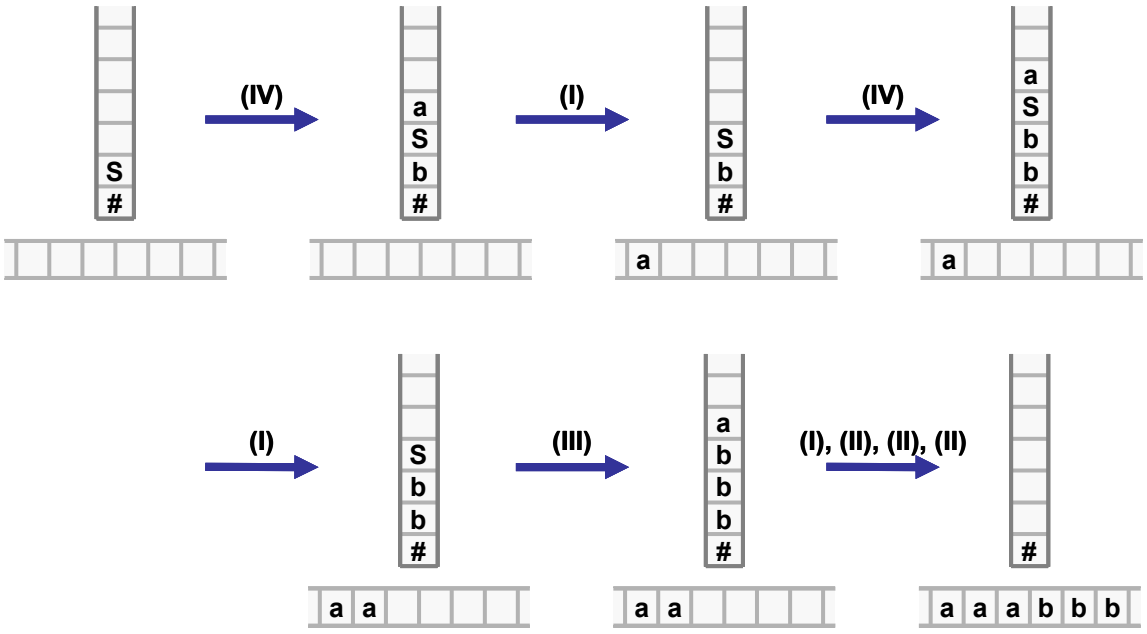


Abbildung 14: Worterzeugung, Beispiel

Die Beispiel für die Anwendung des in Abbildung 13 dargestellten Automaten wird in Abbildung 14 die Erzeugung des Wortes „aaabbb“ dargestellt. Es sind für jeden Schritt sowohl der Zustand des Kellers als auch die aktuelle Ausgabe dargestellt. Die einzelnen Schritte werden von links nach rechts dargestellt, die Pfeile bezeichnen den im jeweiligen Schritt gewählten Übergang.

Die zu Beginn dieses Kapitels aufgestellten Forderungen an Verfahren zur Erzeugung von Wörtern aus Workflow Grammatiken werden von einem generierenden Kellerautomaten, wie er hier vorgestellt wurde, nicht erfüllt.

1. *Wann immer es möglich ist, auf ein Teilwort mindestens eine Regel anzuwenden, dann soll auch eine Regel angewendet werden.*
   Diese Forderung wird nicht erfüllt. Die Möglichkeit, Regeln anzuwenden, wird durch die im Kellerspeicher vorliegenden Nichtterminale repräsentiert. Um eine Regel anwenden zu können, muss also oberste Zeichen des Kellers ein Nichtterminal sein. Terminale werden, sobald sie das oberste Zeichen des Kellers sind, entfernt und auf das Ausgabeband geschrieben. Repräsentiert das oberste Nichtterminal allerdings eine laufende Aktivität, dann ist die Anwendung von Regeln auf dieses Nichtterminal für die Laufzeit der dazugehörigen Aktivität blockiert. Befinden sich weitere nicht blockierte Nichtterminal im Keller, dann kann aber auch auf diese keine Produktionsregel angewendet werden. Der Keller erlaubt Lese- und Schreiboperationen nur auf dem obersten Element. Ein blockiertes Nichtterminal blockiert den gesamten Kellerinhalt und somit auch weitere, zu diesem Zeitpunkt erlaubte Regelanwendungen.
2. *Ist es möglich, Regeln auf verschiedene Teile eines Teilwortes anzuwenden, dann soll frei entscheidbar sein, an welchem Punkt zuerst eine Regel angewendet wird (optionale Forderung).*
   Diese Forderung wird nicht erfüllt. Auch wenn mehrere Nichtterminale im Keller vorliegen, dann können nur auf das oberste Nichtterminal Regeln angewendet werden. Die freie Auswahl des nächsten Nichtterminals ist durch die Einschränkung eines Kellerspeichers nicht möglich.

Ein generierender Kellerautomat ist in der Lage, Wörter einer Grammatik zu erzeugen. Er ist aber, wie soeben dargestellt, nicht anwendbar für die Erzeugung von Wörtern aus einer Workflow Grammatik. Die gestellten Forderungen lassen sich dennoch erfüllen, wenn der Kellerspeicher eines generierenden Kellerautomaten durch ein Arbeitsband, also einen Speicher mit uneingeschränktem Lese- und Schreibzugriff, ersetzt wird. Die Arbeitsweise eines solchen Automaten wird im Folgenden beschrieben.

Zu Beginn steht auf dem Arbeitsband ausschließlich das Startsymbol. In jedem Schritt liest der Automat das aktuelle Teilwort ein und sucht dabei nach Nichtterminalen. Findet er ein oder mehrere (nicht blockierte) Nichtterminale, so wendet er auf genau eines diese Nichtterminale eine Regel an. Dies bedeutet, dass er das Nichtterminal auf dem Arbeitsband mit der rechten Seite der angewendeten Regel überschreibt. Anschließend liest er das so entstandene Teilwort erneut ein und sucht dabei nach weiteren Nichtterminalen. Liegen in einem Teilwort ausschließlich blockierte Nichtterminale vor, so wartet der Automat, bis eines dieser Nichtterminale nicht mehr blockiert ist.

Die zu Beginn dieses Kapitels aufgestellten Forderungen an Verfahren zur Erzeugung von Wörtern aus Workflow Grammatiken werden von dem soeben vorgestellten generierenden Automaten mit einem Arbeitsband statt eines Kellerspeichers erfüllt.

1. *Wann immer es möglich ist, auf ein Teilwort mindestens eine Regel anzuwenden, dann soll auch eine Regel angewendet werden.*
   Das aktuelle Teilwort befindet sich auf dem Arbeitsband. Der Automat hat dadurch jederzeit Zugriff auf das komplette Teilwort. Enthält das Teilwort mindestens ein Nichtterminal, so wird dieses Nichtterminal beim Lesen des Teilwortes entdeckt. Es kann eine Produktionsregel darauf angewendet werden. Blockierte Nichtterminale werden beim Lesen ignoriert, sie beeinflussen nicht den Zugriff auf andere Nichtterminale des Teilwortes. Diese Forderung wird damit erfüllt.
2. *Ist es möglich, Regeln auf verschiedene Teile eines Teilwortes anzuwenden, dann soll frei entscheidbar sein, an welchem Punkt zuerst eine Regel angewendet wird (optionale Forderung).*
   Wie bereits im Bezug auf die erste Forderung dargestellt wurde, hat der Automat jederzeit Zugriff auf das komplette Teilwort. Dementsprechend kann er frei entscheiden, welches der zur Verfügung stehenden Nichtterminale er zuerst verarbeitet. Diese Forderung wird damit ebenfalls erfüllt.

Es wurde damit ein Automat und seine Funktionsweise vorgestellt, mit dem Wörter aus Workflow Grammatiken unter der Beachtung der speziellen Semantik dieser Grammatiken erzeugt werden können. Durch die Ersetzung des Kellerspeichers durch ein Arbeitsband hat sich das Automatenmodell jedoch grundlegend geändert. Der beschriebene Automat ist eine Turingmaschine.

Während gezeigt wurde, dass das allgemeine Erzeugen von Wörtern aus einer kontextfreien Grammatik mit einem Kellerautomaten möglich ist, wurde es durch die speziellen Forderungen von Workflow Grammatiken notwendig, das Modell einer Turingmaschine zu benutzen. Die Komplexität des Verfahrens hat sich also „zwei Klassen nach oben“ bewegt.

Das Automatenmodell einer linear beschränkten Turingmaschine (LBA) wird beim der dieser Arbeit betrachteten Aufgabe des Erzeugens von Wörtern außer Acht gelassen. Eine LBA ist von ihrem Aufbau und ihrer Funktionsweise her zunächst eine vollwertige Turingmaschine. Sie unterliegt jedoch einer Einschränkung bei der Benutzung des Arbeitsbandes. Diese Einschränkung bezieht sich auf die Länge des Eingabewortes. Bei der Erzeugung eines Wortes aus einer Grammatik gibt es jedoch keine Eingabe. Die entsprechenden Verfahren beginnen immer „bei Null“ und erzeugen ein Wort, anstatt eine Eingabe zu verarbeiten. Die Verwendung einer linear beschränkten Turingmaschine zur Erzeugung von Wörtern aus einer Grammatik würde bedeuten, dass eine entsprechende LBA das Arbeitsband nicht benutzen darf. Sie würde damit genau die Mächtigkeit von endlichen Automaten besitzen, da der „Vorteil“ eines Arbeitsbandes verloren geht.

4.3. Kontextsensitive Grammatiken (Chomsky Typ 1)

Die Produktionsregeln Kontextsensitiver Grammatiken ersetzen eine beliebige, nicht leere Folge von Terminalen und Nichtterminalen durch eine wiederum nicht leere Folge von Terminalen und Nichtterminalen. Die einzige Einschränkung lautet dabei, dass ein Teilwort durch die Anwendung einer solchen Produktionsregel nicht kleiner werden darf.

Bei den bisher betrachteten Grammatiktypen hat jedes Nichtterminal eines Teilwortes genau einen Punkt markiert, an dem Regeln auf dieses Teilwort angewendet werden können. Dies war der Fall, weil die Regeln der entsprechenden Grammatiktypen auf der linken Seite immer genau ein Nichtterminal enthalten. Durch diese spezielle Eigenschaft war es beispielsweise möglich, die Aufgabe „Finde alle Möglichkeiten, eine Regel auf dieses Teilwort anzuwenden“ zu reduzieren auf die Aufgabe „Finde alle Nichtterminale in diesem Teilwort“.

Bei kontextsensitiven Grammatiken hat die linke Seite einer Regel im Allgemeinen einen komplexeren Aufbau. Um festzustellen, wo in einem Teilwort Regeln angewendet werden können, ist es nicht mehr ausreichend, nur einzelne Zeichen des Teilwortes zu betrachten. Das Teilwort muss nach Folgen von Terminalen und Nichtterminalen durchsucht werden. Dies bedeutet, dass die im vorherigen Kapitel beschriebene Nutzung eines Kellerautomaten bei der Erzeugung von Wörtern aus kontextsensitiven Grammatiken nicht möglich ist. Ein Kellerautomat hat immer nur Zugriff auf genau ein Zeichen, das oberste Zeichen des Kellerspeichers. Diese Einschränkung verhindert die Betrachtung von Folgen von Zeichen.

Das Erzeugen eines Wortes aus einer allgemeinen kontextsensitiven Grammatik ist ähnlich möglich, wie das im vorherigen Kapitel beschriebene Erzeugen eines Wortes aus einer Workflow Grammatik. Das aktuelle Teilwort steht auf einem Arbeitsband. Der Automat liest in jedem Schritt das komplette Teilwort. Im Gegensatz zu kontextfreien Grammatiken sucht er dabei nicht mehr nur nach Nichtterminalen, sondern nach ganzen Folgen von Terminalen und Nichtterminalen. Findet er mindestens eine der gesuchten Folgen, so kann er darauf eine Regel anwenden, also die Folge aus dem Teilwort entfernen und durch die rechte Seite der entsprechenden Regel ersetzen.

Die zu Beginn dieses Kapitels aufgestellten Forderungen an Verfahren zur Erzeugung von Wörtern aus Workflow Grammatiken werden von dem soeben vorgestellten Verfahren erfüllt. Es entspricht im Wesentlichen dem bereits bei kontextfreien Grammatiken vorgestellten Verfahren zur Erzeugung von Wörtern aus Workflow Grammatiken.

1. *Wann immer es möglich ist, auf ein Teilwort mindestens eine Regel anzuwenden, dann soll auch eine Regel angewendet werden.*

Das aktuelle Teilwort befindet sich auf dem Arbeitsband. Der Automat hat dadurch jederzeit Zugriff auf das komplette Teilwort. Enthält das Teilwort mindestens eine der gesuchten Folgen von Terminalen und Nichtterminalen, so wird diese Folge beim Lesen des Teilwortes entdeckt. Es kann eine Produktionsregel darauf angewendet werden. Blockierte Nichtterminale werden beim Lesen ignoriert, sie beeinflussen nicht den Zugriff auf andere Terminale oder Nichtterminale des Teilwortes. Diese Forderung wird damit erfüllt.

2. *Ist es möglich, Regeln auf verschiedene Teile eines Teilwortes anzuwenden, dann soll frei entscheidbar sein, an welchem Punkt zuerst eine Regel angewendet wird (optionale Forderung).*
   Wie bereits im Bezug auf die erste Forderung dargestellt wurde, hat der Automat jederzeit Zugriff auf das komplette Teilwort. Dementsprechend kann er frei entscheiden, welches der zur Verfügung stehenden Folgen von Terminalen und Nichtterminalen er zuerst verarbeitet. Diese Forderung wird damit ebenfalls erfüllt.

Ein weiterer Unterschied zwischen dem Erzeugen von Wörtern aus kontextfreien Grammatiken und dem Erzeugen von Wörtern aus kontextsensitiven Grammatiken wird im Folgenden noch kurz diskutiert. Bei kontextfreien Grammatiken repräsentiert jedes Nichtterminal in einem Teilwort einen Punkt, an dem Regeln angewendet werden können. Die Menge dieser Punkte verändert sich dabei auf eine ganz bestimmte Art und Weise. Bei jeder Anwendung einer Produktionsregel „verschwindet“ ein solcher Punkt. Werden durch die angewendete Regel andere Nichtterminale erzeugt, so kommen im Gegenzug dazu entsprechend viele „neue“ Punkte dazu, an denen anschließend weitere Regeln angewendet werden können. Die Menge der möglichen Punkte eines Teilwortes, an denen Regeln angewendet werden können, kann also iterativ verwaltet werden. Bei jeder Regelanwendung wird der entsprechende Punkt entfernt und es können neue hinzugefügt werden. Der Rest der Menge bleibt unverändert.

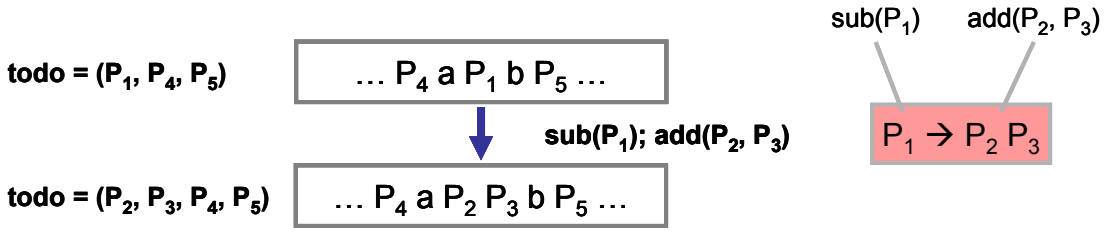


Abbildung 15: Verwaltung der Nichtterminale, kontextfreie Grammatik

Ein Beispiel für dieses Prinzip ist in Abbildung 15 dargestellt. Die Menge „todo“ verwaltet alle Punkte des aktuellen Teilwortes, an denen Regeln angewendet werden können. Bei kontextfreien Grammatiken ist dies immer eine Menge von Nichtterminalen. Jeder Regel einer kontextfreien Grammatik kann, wie im Bild rechts dargestellt, statisch zugeordnet werden, wie sie die Menge „todo“ verändert. Die dargestellt Regel „$P_1 \rightarrow P_2\ P_3$“ bewirkt beispielsweise, dass ein Vorkommen des Nichtterminals P_1 aus der Menge entfernt und jeweils ein Vorkommen von P_2 und P_3 hinzugefügt werden.

Bei kontextsensitiven Grammatiken werden die Punkte eines Teilwortes, an denen Regeln angewendet werden können, durch Folgen von Terminalen und Nichtterminalen repräsentiert. Es ist dabei beispielsweise möglich, dass sich diese Folgen überlappen. Das hat zur Folge, dass bei der Anwendung einer Regel an einem Punkt des Teilwortes die vorher mögliche Anwendung einer Regel auf einen anderen Punkt des Teilwortes unmöglich wird. Umgekehrt ist es möglich, dass durch die Anwendung der gleichen Regel an unterschiedlichen Punkten eines Teilwortes unterschiedlich viele neue Regelanwendungen ermöglicht werden.

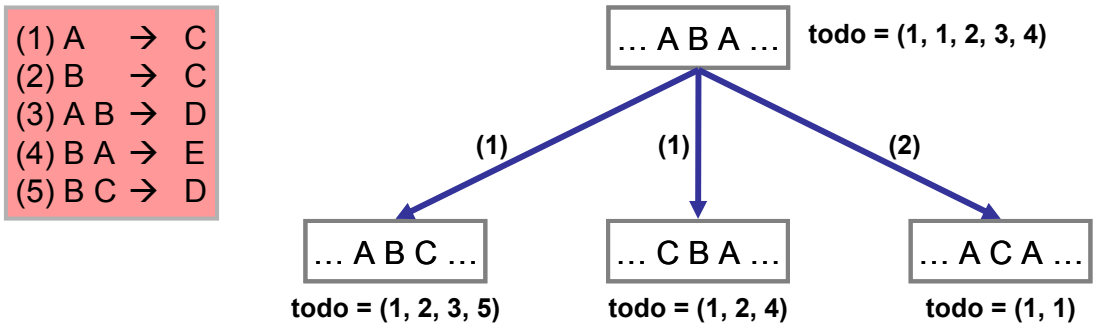


Abbildung 16: Regelanwendung in kontextsensitiven Grammatiken

Ein Beispiel für dieses Verhalten kontextsensitiver Grammatiken ist in Abbildung 16 dargestellt. Die verschiedenen Möglichkeiten, Regeln auf das aktuelle Teilwort anzuwenden, können hier nicht mehr in Form einer Menge von Nichtterminale verwaltet werden. Die Menge „todo“ enthält stattdessen alle Regeln der Grammatik, die im nächsten Schritt angewendet werden können. In der oben dargestellten Ausgangssituation bedeutet dies, dass es zwei Möglichkeiten gibt, die Regel (1) anzuwenden sowie jeweils eine Möglichkeit, die Regeln (2), (3) oder (4) anzuwenden.

Das Beispiel zeigt drei Möglichkeiten, eine Regel auf das Wort „… A B A …“ anzuwenden. In den ersten beiden Varianten wird jeweils die Regel (1) angewendet, allerdings jedes Mal auf ein unterschiedliches Vorkommen des Nichtterminals „A“ innerhalb des Teilwortes. Obwohl die gleiche Regel auf das gleiche Teilwort angewendet wurde, unterscheiden sich die daraus resultierenden „todo“ Mengen. Der Effekt, den eine Regel auf die Menge „todo“ hat, kann also nicht wie bei kontextfreien Grammatiken statisch bestimmt werden. Er ist abhängig vom aktuellen Teilwort und dem Punkt innerhalb des Teilwortes, an dem die Regel angewendet wird.

Diese Betrachtung zeigt, im Vorgriff auf eine mögliche Implementierung der hier diskutierten Verfahren, dass die Komplexität von Verfahren zur Erzeugung von Wörtern aus einer Grammatik in vielerlei Hinsicht vom Typ der verwendeten Grammatik abhängt.

4.4. Rekursiv aufzählbare Grammatiken (Chomsky Typ 0)

Die Produktionsregeln Kontextsensitiver Grammatiken ersetzen eine beliebige, nicht leere Folge von Terminalen und Nichtterminalen durch eine andere Folge von Terminalen und Nichtterminalen. Die Tatsache, dass ein Teilwort einer rekursiv aufzählbaren Grammatik durch die Anwendung einer Produktionsregel kleiner werden kann, hat auf die Erzeugung von Wörtern anhand einer solchen Grammatik keinen Einfluss. Dementsprechend gelten die im vorherigen Kapitel für kontextsensitive Grammatiken getroffen Aussagen gleichzeitig auch für rekursiv aufzählbare Grammatiken.

4.5. Zusammenfassung

In den vorangegangenen Kapiteln wurde für die Grammatiktypen der Chomsky Hierarchie untersucht, wie aus Grammatiken eines bestimmten Typs Wörter erzeugt werden können. Das Erzeugen von Wörtern wurde dabei verglichen mit den bekannten Verfahren zum Erkennen von Wörtern einer Grammatik. Ebenfalls betrachtet wurden dabei die speziellen Anforderungen beim der Erzeugung von Wörtern aus Workflow Grammatiken.

Bei regulären Grammatiken konnte das Modell eines endlichen Automaten syntaktisch unverändert übernommen werden. Ein endlicher Automat einer regulären Grammatik kann verwendet werden, um Wörter genau dieser Grammatik zu erzeugen. Der einzige Unterschied besteht dabei in der Interpretation der Definition des Automaten. Anstatt zu lesen werden Zeichen geschrieben. Dieses Verfahren ist sowohl auf allgemeine reguläre Grammatiken als auch auf reguläre Workflow Grammatiken anwendbar.

Bei kontextfreien Grammatiken wurde ein Verfahren vorgestellt, mit Hilfe eines Kellerautomaten Wörter einer kontextfreien Grammatik zu erzeugen. Auch in diesem Fall konnte also das Automatenmodell des entsprechenden Akzeptors übernommen werden. Die Anwendung auf kontextfreie Workflow Grammatiken war in diesem Fall aber nicht möglich. Für die Erzeugung von Wörtern aus einer kontextfreien Workflow Grammatik wurde der Kellerautomat zu einer Turingmaschine erweitert.

Sowohl bei kontextsensitiven als auch bei rekursiv aufzählbaren Grammatiken wurde das gleiche Verfahren zum Erzeugen von Wörtern beschrieben. Im Gegensatz zum Erkennen von Wörtern kann auf die beiden unterschiedlichen Grammatiktypen jeweils das gleiche Verfahren angewendet haben. Die Unterschiede zwischen kontextsensitiven und rekursiv aufzählbaren Grammatiken sind für das Erzeugen von Wörtern nicht relevant.

5 Aufbau einer Workflow Engine

In den folgenden Kapiteln 5, 6 und 7 wird der allgemeine Aufbau sowie die Funktionsweise einer Workflow Engine betrachtet (Kapitel 5), anschließend eine Workflow Engine zur Ausführung von Workflow Grammatik Prozessmodellen in einer Cloud Umgebung entworfen (Kapitel 6) und diese dann prototypisch implementiert (Kapitel 7). Der Begriff einer Workflow Engine wird im Folgenden zunächst einmal definiert und zusätzlich abgegrenzt von dem Begriff eines Workflow Management Systems. Die folgenden Definitionen orientieren sich an den von der Workflow Management Coalition (WfMC)[5] definierten und verwendeten Begriffen [WFMC99].

Workflow Management System

Ein *Workflow Management System* besteht aus verschiedenen Softwarekomponenten. Diese Komponenten können:

- Prozessdefinitionen speichern und interpretieren
- Workflow Instanzen erzeugen und während ihrer Ausführung verwalten
- Die Interaktion der Workflow Instanzen mit Workflow Teilnehmern und Anwendungen kontrollieren

Ein Workflow Management System ist damit ein Softwaresystem, welches es ermöglicht, (Geschäfts-) Prozesse in einer IT Umgebung zu definieren und auszuführen. Es besteht aus einer Menge von Softwarekomponenten, welche die dafür benötigten Funktionalitäten implementieren. Eine dieser Komponenten ist die Workflow Engine.

Workflow Engine

Eine *Workflow Engine* ist ein Software Dienst oder auch eine „Maschine“, welche die Laufzeitumgebung für eine Prozessinstanz bietet. Die Funktionen einer Workflow Engine umfassen unter anderem:

- Die Interpretation der Prozessdefinition
- Das Erzeugen von Prozessinstanzen und die Verwaltung ihrer Ausführung (starten, beenden, pausieren, fortsetzen,…)
- Die Navigation zwischen Aktivitäten und der Aufruf ihrer Implementierung bzw. die Erzeugung entsprechender Workitems

Kapitel 0 beschreibt den Aufbau von Workflow Management Systemen am Beispiel unterschiedlicher Implementierungen und Produkte. Für jedes System werden die wesentlichen Komponenten und ihre Funktion identifiziert und kurz beschrieben.

In Kapitel 5.2 die zuvor beschriebenen Architekturen mit dem Fokus auf die Prozessausführung durch die Workflow Engine zusammen gefasst.

Kapitel 5.3 leitet aus den vorhergegangenen Betrachtungen die wesentlichen Komponenten einer Workflow Engine ab und beschreibt diese allgemein.

In Kapitel 5.4 wird an zwei beispielhaften Anwendungsszenarien gezeigt, wie die in den vorherigen Kapiteln hergeleitete Modularität einer Workflow Engine innerhalb einer Cloud Umgebung ausgenutzt werden kann.

[5] http://www.wfmc.org/

5.1. Überblick über bestehende Architekturen

Im Folgenden soll zunächst ein Überblick über mögliche Architekturen von Workflow Systemen gegeben werden. Das in Kapitel 5.1.1 beschriebene Referenzmodell der Workflow Management Coalition stellt dabei eine Besonderheit dar. Es beschreibt eine abstrakte Architektur, es basiert nicht auf einer konkreten Implementierung. Alle weiteren vorgestellten Systemarchitekturen beschreiben dagegen den Aufbau bestehender Softwaresysteme.

5.1.1. Workflow Management Coalition – The Workflow Reference Model

Die *Workflow Management Coalition (WfMC)* ist ein 1993 gegründeter Zusammenschluss von Anwendern, Entwicklern, Beratern, Analysten und Universitäts- sowie Forschungsgruppen, die im Bereich von Workflow und *Business Process Management (BPM)* aktiv sind[6].

Die Workflow Management Coalition hat 1995 das sogenannte *Workflow Reference Model* vorgestellt. Die Definition dieses Modells war dadurch motiviert, dass es den bestehenden Workflow Management Produkten allgemein nicht möglich war, miteinander zu interagieren. Das Workflow Management Coalition Reference Model soll einen allgemeingültigen Standard definieren, welcher die Interoperabilität heterogener Workflow Management Systeme ermöglicht [Hol95].

Die Komponenten und Schnittstellen des Workflow Reference Models sind in Abbildung 17 dargestellt. Die einzelnen Komponenten sowie die Schnittstellen werden im Folgenden kurz vorgestellt.

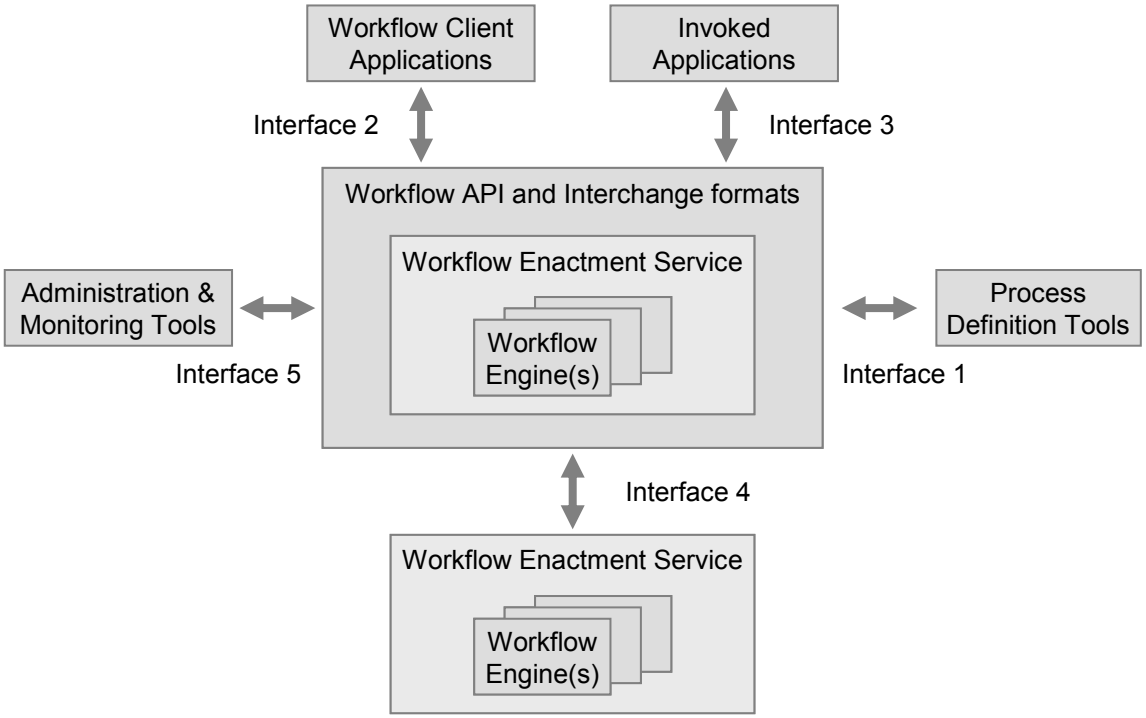


Abbildung 17: Workflow Management Coalition Reference Model

5.1.1.1. Komponente Workflow Enactment Service

[6] http://www.wfmc.org/about-us.html

Der *Workflow Enactment Service* bildet die Laufzeitumgebung, innerhalb welcher Prozessinstanzen ausgeführt werden. Die Ausführung erfolgt dabei nicht durch den Workflow Enactment Service selbst, sondern durch die im Workflow Enactment Service enthaltenen Workflow Engine Komponenten. Der Workflow Enactment Service bietet die im Folgenden beschriebenen Schnittstellen 1 bis 5 an.

5.1.1.2. Komponente Workflow Engine

Eine *Workflow Engine* verwaltet den Lebenslauf von Prozessinstanzen. Sie erzeugt, startet und beendet einzelne Prozessinstanzen. Die Workflow Engine interpretiert die durch die Prozessdefinition vorgegebene Prozessstruktur und navigiert dementsprechend zwischen den einzelnen Prozessaktivitäten. Sie nutzt dabei die in Abbildung 17 dargestellten Schnittstellen 2 und 3, um auf die Implementierungen der Aktivitäten (externe Anwendungen) zuzugreifen.

Die Daten jeder Prozessinstanz werden ebenfalls von der Workflow Engine verwaltet. Es wird dabei zwischen zwei Datentypen unterschieden. *Workflow Prozessdaten* sind die durch die Prozessdefinition vorgegebenen Datenelemente, welche von der Prozesslogik benötigt und verwendet werden (die Variablen des Prozesses). Als *Workflow Kontrolldaten* werden die von der Workflow Engine selbst definierten und benötigten Daten für jede Prozessinstanz bezeichnet. Dies sind beispielsweise Informationen über den Prozesszustand oder über den Zustand einzelner Prozessaktivitäten. Workflow Prozessdaten und Workflow Kontrolldaten repräsentieren gemeinsam den Zustand einer Prozessinstanz.

5.1.1.3. Schnittstelle 1, Process Definition Tools

Die Prozessmodelle, welche vom Workflow Management System ausgeführt werden, müssen zunächst modelliert und anschließend auf dem Workflow Management System installiert werden. Die Modellierung geschieht über sogenannte *Process Definition Tools*, also ganz allgemein gesagt mit Hilfe von Modellierungswerkzeugen. Die Installation eines Prozessmodells auf dem Workflow Management System erfolgt über die Schnittstelle 1. Sie erlaubt sowohl den Import von Prozessmodellen in das Workflow Management System hinein als auch den Export von Prozessmodellen aus dem Workflow Management System hinaus.

5.1.1.4. Schnittstelle 2, Workflow Client Applications

Die Schnittstelle zu *Workflow Client Applications* realisiert den Aufruf von Aktivitäten, die Nutzerinteraktion beinhalten. Solche Aktivitäten basieren auf dem Konzept der *Workitems* und der *Worklist Handler*. Die Workflow Engine erzeugt bei der Aktivierung einer entsprechenden Aktivität ein Workitem. Diese Workitem enthält Informationen darüber, welcher Nutzer welche Aktivität mit welcher Ressource durchführen soll. Ein Workitem könnte also beispielsweise aussagen, dass ein Finanzsachbearbeiter (wer) eine Bonitätsprüfung (was) mit Hilfe von Excel und einer Datenbank (womit) durchführen soll. Die Workitems werden dann von den Nutzern über ihre jeweiligen Worklist Handler aktiviert, abgearbeitet und abgeschlossen.

5.1.1.5. Schnittstelle 3, Invoked Application Functions

Aktivitäten, die durch Programme implementiert werden, ruft die Workflow Engine über die Schnittstelle *Invoked Application Functions* auf. Die Schnittstelle bietet die Möglichkeit, die Komplexität eines beliebigen Programmaufrufs von der Workflow Engine zu entkoppeln. Dazu bietet sie standardisierte Schnittstellen für beliebige Programmaufrufe sowie standardisierte Datenformate an. Der eigentliche Programmaufruf wird dann von einer weiteren Komponente übernommen, welche die meist nötige Konvertierung der Daten übernimmt. Eine mögliche Ausprägung einer solchen generischen Programmaufrufschnittstelle stellen Web Services dar, wie sie beispielsweise von BPEL

und auch von Workflow Grammatiken genutzt werden. Eine weitere mögliche Form der Schnittstelle Invoked Application Functions ist der direkte Aufruf der entsprechenden Implementierung. Dies setzt jedoch unter anderem voraus, dass die Workflow Engine das entsprechende Aufrufparadigma implementiert sowie das Datenformat der Implementierung kennt.

5.1.1.6. Schnittstelle 4, Workflow Interoperability

Das Workflow Reference Model stellt Szenarien vor, in denen mehrere Workflow Management Systeme kooperativ zusammen arbeiten. Neben der Verteilung eines einzelnen Prozesses auf mehrere Workflow Management Systeme ist auch die Zusammenarbeit (Choreographie) von mehreren Prozessen, die auf jeweils unterschiedlichen Workflow Management Systemen ausgeführt werden, ein vorgestellter Anwendungsfall.

Um die Zusammenarbeit von potentiell unterschiedlichen Workflow Management Systemen zu ermöglichen, wurde die Schnittstelle *Workflow Interoperability* definiert. Sie erlaubt es, dass Workflow Management Systeme in heterogenen Umgebungen auf einheitliche Weise zusammen arbeiten können.

5.1.1.7. Schnittstelle 5, Administration & Monitoring

Die Schnittstelle *Administration & Monitoring* bietet die Möglichkeit, das Workflow Management System zu verwalten dun zu überwachen. Die Administration umfasst beispielsweise die Benutzerverwaltung, Auditingkonfiguration oder die Verwaltung der installierten Prozessmodelle (beispielsweise Versionierung oder De-/Aktivierung). Das Monitoring bietet die Möglichkeit, den Zustand aller aktiven Prozesse und der damit assoziierten Daten abzufragen und zu überwachen.

5.1.2. Apache ODE

Apache ODE (Orchestration Director Engine) ist ein Workflow Management System für die Ausführung von BPEL Prozessmodellen (es werden sowohl BPEL4WS 1.1 als auch WS-BPEL 2.0 Prozessmodelle unterstützt). Apache ODE wird als quelloffene Software von der Apache Software Foundation entwickelt[7].

Die Ausführung von BPEL Prozessen erfolgt in Apache ODE durch das *JACOB Framework*. Dieses Framework erlaubt die parallele Ausführung von Aktivitäten sowie die persistente Speicherung des aktuellen Ausführungszustandes. Während der Ausführung eines Prozesses werden die betreffenden BPEL Aktivitäten auf ausführbare JACOB Objekte abgebildet und von der JACOB Engine ausgeführt. Der Kontrollfluss zwischen diesen Objekten wird durch sogenannte *Channel* realisiert. Sie verbinden die einzelnen Objekte miteinander und leiten den Kontrollfluss weiter.

In Abbildung 18 wird der konzeptionelle Aufbau von Apache ODE beschrieben. BPEL Prozessmodelle werden bei der Installation zunächst in eine interne Repräsentation überführt. Die JACOB Engine, welche die eigentliche Prozessausführung implementiert, befindet sich innerhalb der *ODE BPEL Runtime* Umgebung. Diese Runtime Umgebung bietet eine Management Schnittstelle nach außen an.

[7] http://ode.apache.org

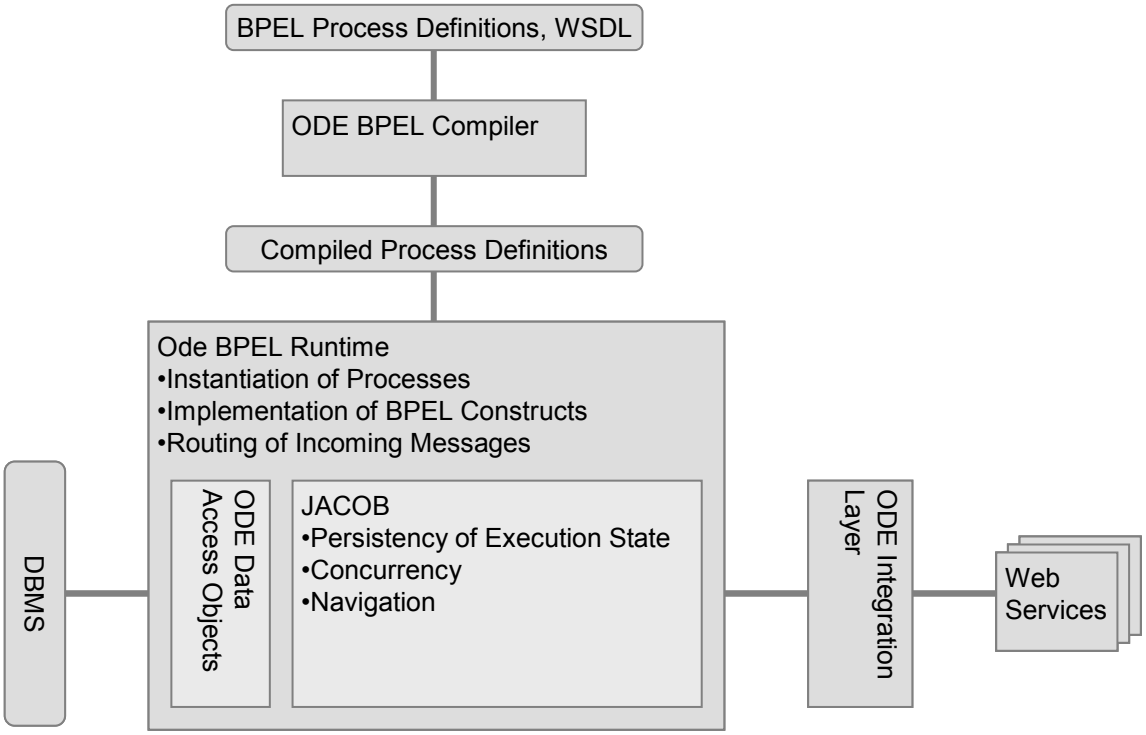


Abbildung 18: Apache ODE

Die persistente Speicherung der Prozessdaten erfolgt über *Data Access Objects (DAOs)* in eine darunterliegende Datenbank. Apache ODE wird standardmäßig mit einer internen Datenbank installiert, lässt sich jedoch auch so konfigurieren, dass andere Datenbanken genutzt werden können. Der Aufruf von Web Services erfolgt über den *ODE Integration Layer*. Dieser implementiert die konkreten Zugriffsmechanismen, also spezifische Protokolle und Nachrichtenformate, und koppelt sie so von der JACOB Engine ab.

5.1.3. Workflow Execution Engine (WEE)

Das Ziel beim Entwurf der *Workflow Execution Engine (WEE)* war der Aufbau einer flexiblen Workflow Engine unter der Nutzung serviceorientierter Ansätze[8]. Die zentrale WEE Komponente ist für die Interpretation eines Prozessmodells sowie die Verfolgung des Kontrollflusses aktiver Prozessinstanzen verantwortlich. Sie ist nicht verantwortlich für den Aufruf von Prozessaktivitäten, die Verwaltung von Ressourcen, das Monitoring oder die Behandlung von Ausnahmen während der Prozessausführung [Stu10].

Zur besseren Verständlichkeit wird die Workflow Execution Engine selbst im Folgenden als *WEE Komponente* bezeichnet. Das komplette System, bestehend aus *WEE Komponente*, *Handler Wrapper* und den *Handlern* wird als *WEE System* bezeichnet.

Abbildung 19 zeigt den prinzipiellen Aufbau von des WEE Systems. Fähigkeiten, welche die WEE Komponente selbst nicht implementiert, die für die Ausführung von Prozessmodellen aber notwendig (Aufruf von Aktivitäten) oder erwünscht sind (Monitoring), werden über den Handler Wrapper integriert. Die Integration erfolgt über mehrere vordefinierte externe Schnittstellen. Über diese Schnittstellen können sogenannte Handler in das WEE System integriert werden.

[8] http://www.wst.univie.ac.at/workgroups/wee

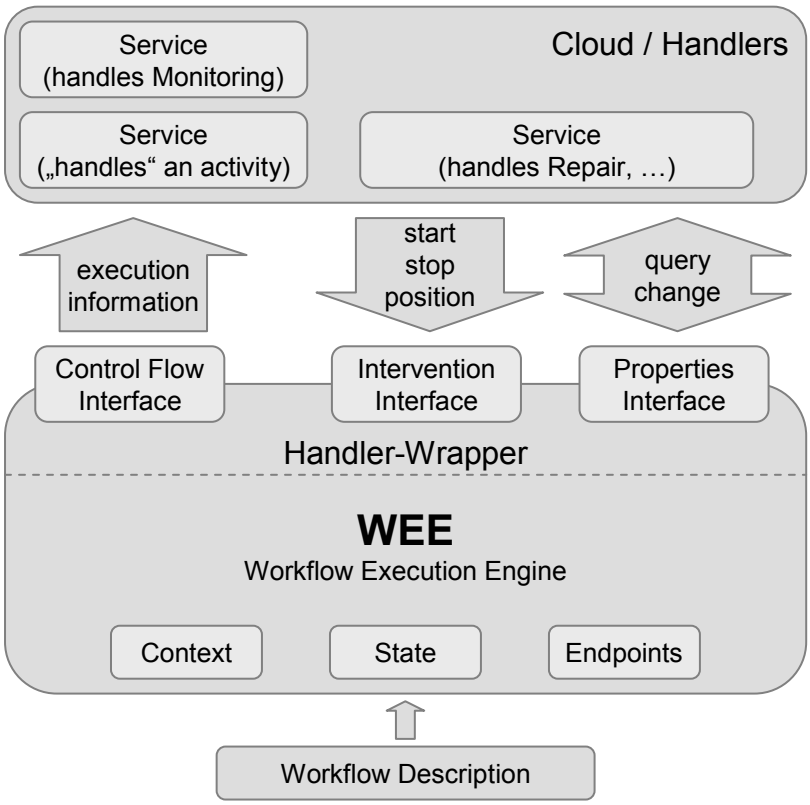


Abbildung 19: Workflow Execution Engine (WEE)

Die Schnittstellen des Handler Wrappers erlauben sowohl die Aktivierung von Handlern durch die WEE Komponente, als auch die Steuerung und Manipulation der WEE Komponente durch die Handler. Der Handler Wrapper entkoppelt dabei die WEE Komponente von den Handlern. Die über den Handler Wrapper registrierten Handler implementieren auf diesem Weg alle vom WEE System benötigten oder erwünschten zusätzlich Funktionalitäten.

5.1.4. Oracle BPEL Process Manager

Der *Oracle BPEL Process Manager* ist ein von der Firma Oracle[9] vertriebenes Workflow Management System zur Ausführung von BPEL Workflows. Der grundlegende Aufbau des Oracle BPEL Process Managers ist in Abbildung 20 dargestellt. Das gesamte System läuft auf einem *J2EE Application Server*. Die Entwicklung und Installation von Prozessmodellen erfolgt über den *BPEL Designer*. Verwaltet wird der Oracle BPEL Process Manager über die sogenannte *BPEL Konsole* [JMS06].

Die Kernkomponente für die Ausführung von Workflows ist der *BPEL Server*. Die darin enthaltene *Core BPEL Engine* implementiert die eigentliche Ausführung der Prozesse. Sie greift dabei auf die anderen Komponenten des BPEL Servers zurück.

[9] http://www.oracle.com/de/index.html

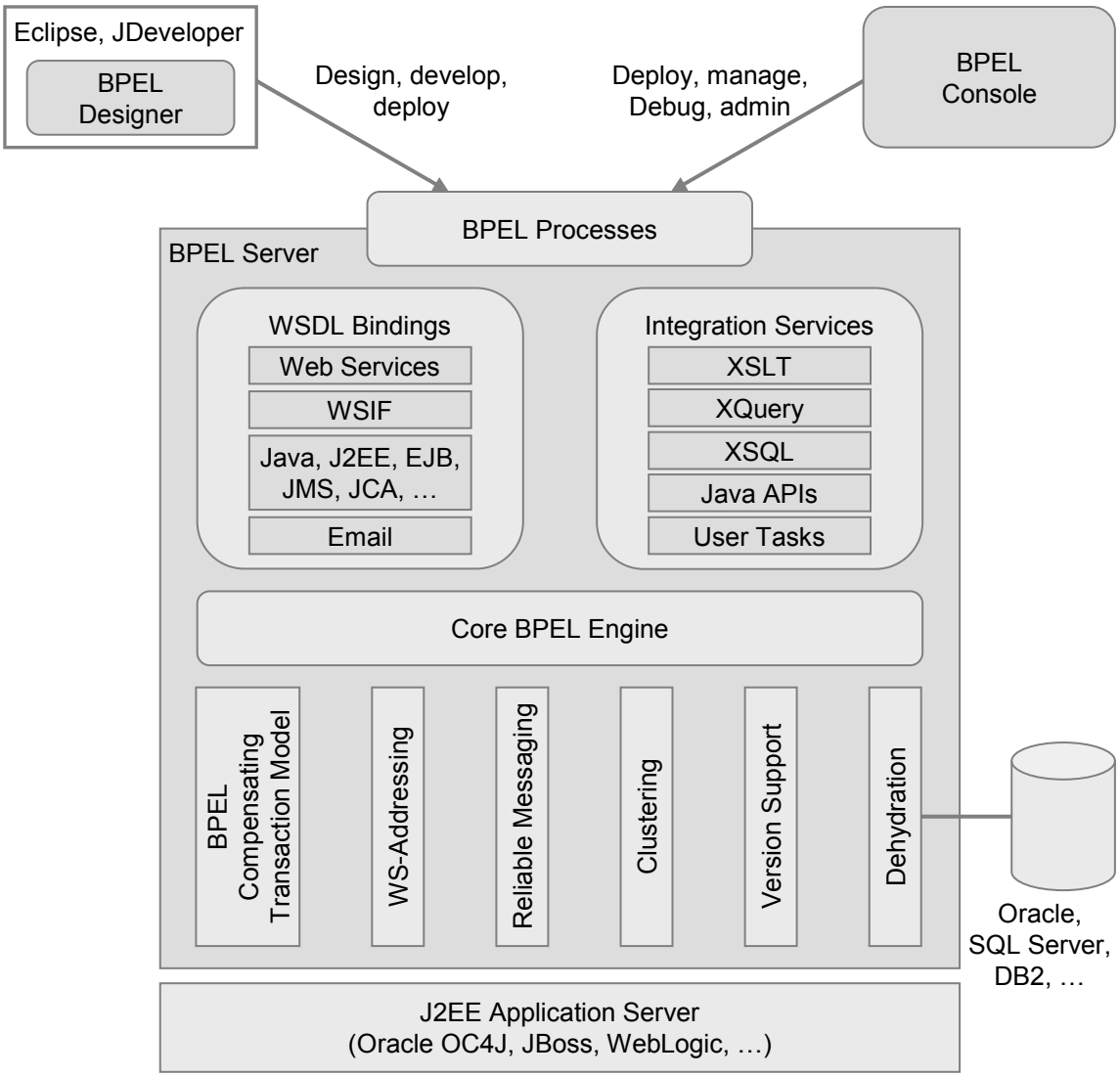


Abbildung 20: Oracle BPEL Process Manager

Die Komponente *Dehydration* realisiert die persistente Speicherung des jeweils aktuellen Zustandes einer Prozessinstanz in einer Datenbank. Die Komponente *WSDL Bindings* realisiert Web Service Kommunikation, wie im BPEL Standard beschrieben [BPEL07]. Sie bietet die Schnittstellen, über die ein Prozess von außen aufgerufen werden kann und wird umgekehrt von der Core BPEL Engine verwendet, um während der Prozessausführung weitere Web Services aufzurufen. Die Komponente *Integration Services* implementiert Funktionalitäten, die im BPEL Standard nicht beschrieben sind. Dies sind beispielsweise die Nutzung von XSLT Transformationen oder der direkte Aufruf von Java Objekten.

5.1.5. YAWL Workflow System

YAWL (Yet Another Workflow Language) bezeichnet sowohl eine Prozessbeschreibungssprache als auch ein Workflow System zur Ausführung von YAWL Prozessmodellen. Die Sprache YAWL entstand aus einer ausführlichen Analyse bestehender Prozessbeschreibungssprachen und Workflow Management Systeme [AH05]. Basierend auf dieser Analyse wurde ein Katalog von Workflow Mustern erstellt, welcher die beobachteten Modellierungskonstrukte abstrahiert, klassifiziert und im

Detail beschreibt[10]. Als Basis für die einheitliche Modellierung aller dieser Konstrukte wurden Petri-Netze gewählt [PW08]. Die daraus entstandene Sprache YAWL beschreibt Prozesse mit Hilfe von erweiterten Petri-Netzen.

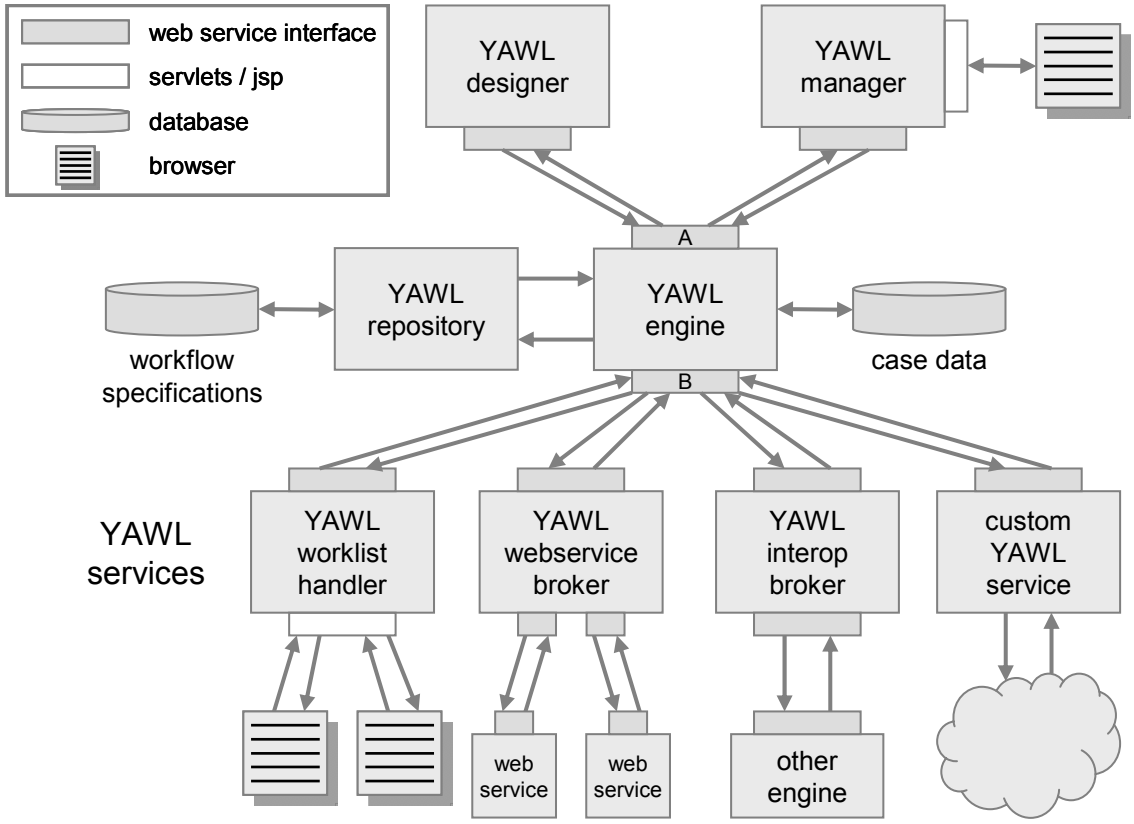


Abbildung 21: YAWL Workflow System

Zusammen mit der Sprache YAWL wurde ein Workflow Management System zur Ausführung von YAWL Prozessmodellen entwickelt [AAD+04]. Die Architektur dieses Systems ist in Abbildung 21 dargestellt. Die Verwaltung des YAWL Workflow Systems erfolgt über den *YAWL Manager*. YAWL Prozessmodelle werden mit dem *YAWL Designer* modelliert und anschließend auf der *YAWL Engine* installiert. Die installierten Modelle werden durch das *YAWL Repository* in einer Datenbank abgelegt. Instanzen von Prozessmodellen werden im YAWL Workflow System als *Case* bezeichnet. Die YAWL Engine instanziiert Prozessmodelle und ist verantwortlich für deren Durchführung. Die Zustandsdaten aktiver Prozessinstanzen werden dabei von der YAWL Engine verwaltet und während der Ausführung in einer ihr zugeordneten Datenbank gespeichert.

Die Interaktion zwischen der YAWL Engine und ihrer Umgebung findet über sogenannte *YAWL Services* statt. Der *YAWL Worklist Handler* implementiert das bereits in Kapitel 5.1.1.4 beschriebene Konzept von Aktivitäten mit Nutzerinteraktion über Workitems und Worklists. Der *YAWL Web Service Broker* dient als Schnittstelle für Web Service Kommunikation. Die YAWL Engine kann über dieses Modul zum einen Web Services aufrufen und zum anderen selbst Web Service Schnittstellen nach außen hin anbieten. Der *YAWL Interoperability Broker* erlaubt es, die YAWL Engine mit anderen Workflow Systemen zu integrieren. Neben diesen drei YAWL Services ist es möglich, das YAWL Workflow System um beliebige weitere YAWL Services zu erweitern. Dies wird in der Abbildung durch den *Custom YAWL Service* symbolisiert.

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

5.2. Zusammenfassung der vorgestellten Workflow Management Systeme

Die folgende Zusammenfassung der in den vorherigen Abschnitten vorgestellten Workflow Management Systeme konzentriert sich auf den Aufbau der jeweiligen Workflow Engine, also derjenigen Komponente, die nach der zu Beginn vorgestellten Definition wesentlich für die Ausführung eines Prozesses verantwortlich ist. Komponenten oder Funktionalitäten wie Monitoring, Administration oder Prozessmodellierung sind zwar sinnvolle und nötige Bestandteile eines Workflow Management Systems, sie sind aber für die eigentliche Ausführung eines Prozesses nicht notwendig.

Workflow Reference Model

Das Modell der Workflow Management Coalition definiert eine allgemeine Architektur eines Workflow Management Systems mit dem Schwerpunkt auf externe Schnittstellen. Das Modell definiert dabei unter anderem die Komponente „Workflow Engine“, sie ist Teil eines Workflow Enactment Services und führt Prozessmodelle aus. Die Workflow Engine verwaltet dabei zwei Arten von Daten, die Workflow Prozessdaten (Variablen) sowie die Workflow Kontrolldaten (Prozesszustand). Zur Ausführung von Prozessaktivitäten nutzt die Workflow Engine die Schnittstellen „Workflow Client Applications“ (Workitem Konzept) sowie „Invoked Application Functions“ (Programmaufrufe).

Apache ODE

Die zentrale Ausführungskomponente der Apache ODE ist die JACOB Engine. Sie verwaltet den Kontrollfluss konkreter Prozessinstanzen. Die Persistierung von Daten (sowohl Prozessvariablen als auch Zustandsdaten) erfolgt über eine DAO Schicht in einem Datenbanksystem. Die Kommunikation nach außen, und damit auch der Aufruf von Aktivitäten, erfolgt über den ODE Integration Layer.

Workflow Execution Engine (WEE)

Zentraler Bestandteil eines WEE Systems ist die WEE Komponente. Sie koordiniert die Ausführung von Prozessinstanzen. Zusätzlich verwaltet sie die dazu benötigten Daten, sie werden unterschieden in Kontext (Prozessvariablen), Zustand und Endpunkte. Die Ausführung von Aktivitäten wird von der WEE Komponente an den Handler Wrapper delegiert und von den dort registrierten Handlern realisiert. Diese Handler sind von der WEE Komponente entkoppelt. Sie erlauben es zudem, die WEE Komponente dynamisch um zusätzliche Funktionalitäten, wie beispielsweise Monitoring oder erweiterte Fehlerbehandlung, zu erweitern.

Oracle BPEL Process Manager

Die zentrale Ausführungskomponente des Oracle BPEL Process Managers ist die Core BPEL Engine, sie navigiert durch BPEL Prozessmodelle. Alle benötigten Datenzugriffe erfolgen über die Dehydration Komponente auf eine Datenbank. Der Aufruf von Aktivitäten basiert auf Web Services und verwendet mehrere Module, das wesentliche sind dabei die Integration Services. Weitere Module, wie WS-Addressing oder WSDL Bindings unterstützen die Web Service Kommunikation oder realisieren, wie beispielsweise die Clustering Komponente, nichtfunktionale Eigenschaften (Quality of Services, QoS).

YAWL

Die zentrale Ausführungskomponente des YAWL Systems ist die YAWL Engine. Der Zugriff auf die Prozessmodelldefinition erfolgt über das YAWL Repository. Der Zugriff auf die Instanzdaten (Variablen und Ausführungszustand) erfolgt direkt auf eine Datenbank (case data). Für den Aufruf von Aktivitäten unterstützt YAWL zwei Konzepte. Der Aufruf von Aktivitäten mit Benutzerinteraktion erfolgt über den YAWL Worklist Handler. Der Aufruf von Aktivitäten, die durch Programme implementiert werden, erfolgt über den YAWL Web Service Broker. Der Aufbau des YAWL Wokflow Systems folgen im Wesentlichen dem bereits beschriebenen Workflow Reference Model der Workflow Management Coalition.

5.3. Komponenten einer Workflow Engine

Die Idee, eine Workflow Engine modular aufzubauen und ihre Funktionalitäten in einzelne Komponenten zu untergliedern, ergibt allein schon aus der Tatsache, dass eine Workflow Engine im Allgemeinen ein komplexes Softwaresystem darstellt [Bal09], [PCW85]. Diesbezügliche Anforderungen an Workflow Engines und die sich daraus ergebenden Schlüsse für deren Architektur, werden unter anderem in [GHS95] ausführlicher diskutiert.

Eine genauere Betrachtung der in den vorigen Abschnitten betrachteten Workflow Management Systeme und der entsprechenden Workflow Engines ergibt, dass alle vorgestellten Systeme ähnliche Komponenten besitzen. Im Folgenden wird aus dieser Beobachtung der generische Aufbau einer Workflow Engine abgeleitet. Für die einzelnen Komponenten wird motiviert, warum sie allgemeingültiger Bestandteil einer Workflow Engine sind. Zum anderen wird ihre jeweilige Funktion innerhalb der Workflow Engine näher erläutert.

5.3.1. Navigator

Der Navigator ist die zentrale Komponente jeder Workflow Engine. Er ist dafür verantwortlich, den Kontrollfluss aktiver Prozessinstanzen zu verfolgen. Der Navigator bestimmt anhand des Prozessmodells sowie des aktuellen Zustands einer Prozessinstanz, welche Aktivitäten aktiviert werden können (und er aktiviert sie dann auch). Der Navigator benötigt für die Ausführung weitere Komponenten. Im Folgenden werden diese Komponenten sowie ihr Zusammenspiel mit dem Navigator beschrieben.

Der Begriff des Navigators leitet sich aus der Ausführung graphbasierter Prozessmodelle ab. Die Verfolgung des Kontrollflusses bedeutet bei solchen Modellen, innerhalb eines Graphen zu bestimmen, welche Kanten bzw. Pfade verfolgt werden. Der Navigator „bewegt“ sich also durch den Prozessgraphen und bestimmt dabei, wo es langgeht, er navigiert.

5.3.2. Prozessmodellspeicher

Prozessinstanzen sind Instanzen eines Prozessmodells. Der Navigator kontrolliert die Ausführung von Prozessinstanzen anhand des ihnen zugrunde liegenden Prozessmodells. Das bedeutet, dass die entsprechenden Prozessmodelle dem Navigator verfügbar gemacht werden müssen.

Abbildung 22 demonstriert das Zusammenspiel des Navigators mit dem Prozessmodellspeicher an einem Beispiel. Es ist ein einfaches Prozessmodell, bestehend aus den Aktivitäten X_1, X_2 und X_3, dargestellt. Es wird angenommen dass die Aktivität X_1 soeben beendet worden ist. Um zu bestimmen, welche Aktivität bzw. welche Aktivitäten als nächsten zu aktivieren sind, startet der Navigator eine entsprechende Anfrage an den Prozessmodellspeicher. Der Prozessmodellspeicher gibt im dargestellten Fall zurück, dass die Aktivitäten X_2 und X_3 als nächstes aktiviert werden sollen, was der Navigator dann auch tut (wie genau er dies tun kann, wird in Kapitel 5.3.5 diskutiert).

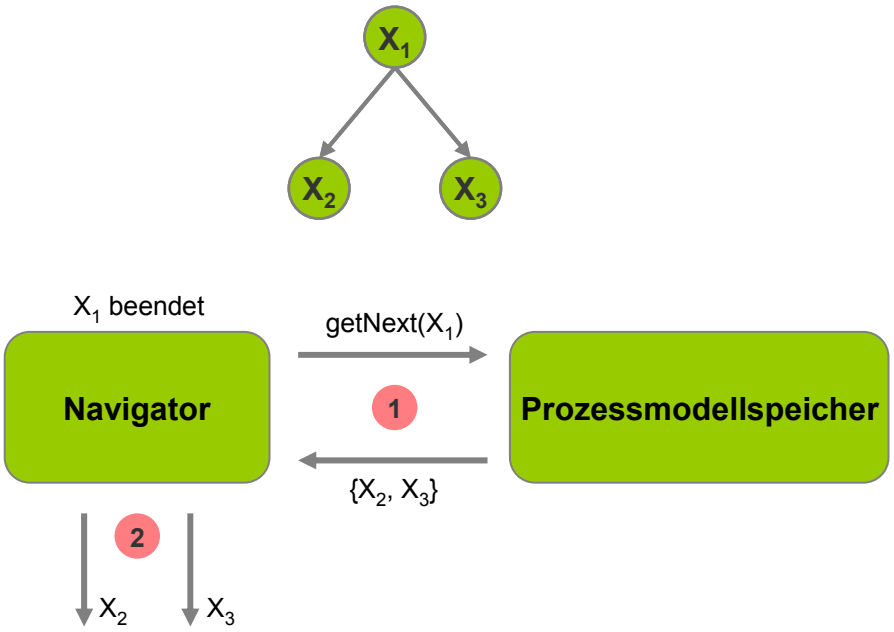


Abbildung 22: Verwendung Prozessmodellspeicher

Im dargestellten Beispiel leistet der Prozessmodellspeicher mehr als nur die Bereitstellung des Prozessmodells. Die Anfrage „getNext(…)“ beinhaltet, dass der Prozessmodellspeicher das Prozessmodell versteht, denn nur wenn er dies tut, kann er die Nachfolger einer Aktivität bestimmen. Ein einfacher aufgebauter Prozessmodellspeicher würde dem Navigator lediglich das Prozessmodell übergeben. Der Navigator wäre dann selbst dafür verantwortlich, beispielsweise die Nachfolgeaktivitäten zu bestimmen. Der Prozessmodellspeicher kann also in verschiedenen Ausprägungen vorliegen. Er muss mindestens Speicherfunktionalität anbieten, kann aber auch noch zusätzliche Funktionalitäten zur Verfügung stellen.

Für die Ausführung von Prozessinstanzen ist lediglich relevant, dass die entsprechenden Prozessmodelle verfügbar sind. Die Verwaltung der Prozessmodelle, also beispielsweise die Installation, die Aktivierung- oder Deaktivierung oder die Versionierung, ist Aufgabe des Workflow Management Systems.

5.3.3. Instanzspeicher

Prozessmodelle können neben dem Kontrollfluss und den daran beteiligten Aktivitäten auch Daten spezifizieren. Diese werden mit dem Prozess assoziiert, die Aktivitäten des Prozesses können darauf lesend und schreibend zugreifen. Diese Prozessdaten werden, in Anlehnung an die Begriffe klassischer Programmiersprachen, im Folgenden auch als Variablen bezeichnet. Variablen werden während der Ausführung einer Prozessinstanz instanziiert, erhalten konkrete Werte, können verändert werden, und werden spätestens zum Ende der Prozessinstanz wieder gelöscht.

Die Verwendung des Speichers für Prozessdaten wird in Abbildung 23 an einem Beispiel gezeigt. Der Navigator hat die Aktivität X_1 aktiviert. Die Aktivität benötigt lesenden Zugriff auf die Variablen V_1 und V_2 (Eingabedaten) sowie schreibenden Zugriff auf die Variable V_1 (Ausgabedaten). Bevor der Navigator die Aktivität ausführen kann, fragt er beim Prozessdatenspeicher die aktuellen Werte der Variablen V_1 und V_2 ab (1). Diese Werte werden dann der für die Ausführung der Aktivität zuständigen Komponente übergeben (2). Die Ausführung von Aktivitäten wird in Kapitel 5.3.5 näher betrachtet. Am Ende der Ausführung der Aktivität X_1 erhält der Navigator ein Ergebnis zurück und speichert den neuen Wert der Variablen V_1 im Prozessdatenspeicher ab (3).

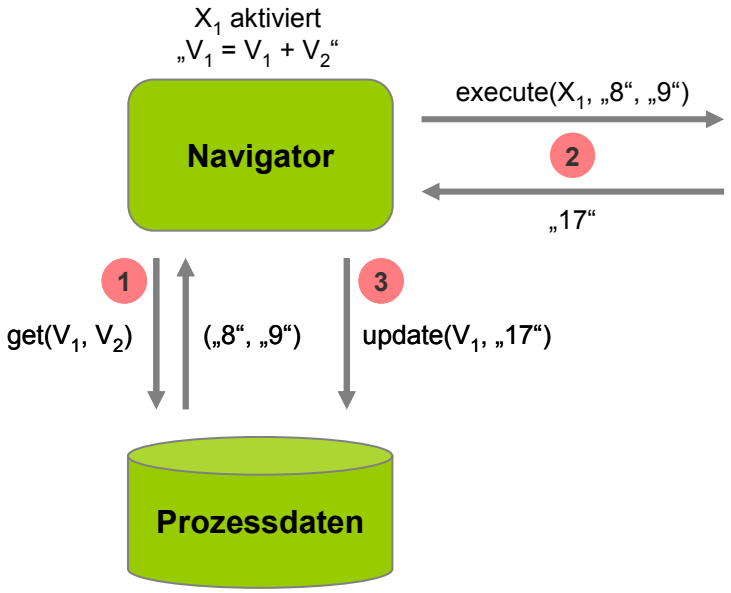


Abbildung 23: Prozessdatenspeicher

Der Zustand einer Prozessinstanz ist durch die mit der Prozessinstanz assoziierten Variablen (Prozessdaten) noch nicht eindeutig und vollständig beschrieben. Zur vollständigen Beschreibung des Zustandes einer Prozessinstanz gehören zusätzlich Informationen darüber, welche Aktivitäten beendet wurden, in welchem Zustand sie beendet wurden (beispielsweise „erfolgreich“ oder „fehlerhaft“) oder welche Aktivitäten aktuell aktiv sind. Diese Daten werden im Folgenden als Prozessausführungszustand bezeichnet.

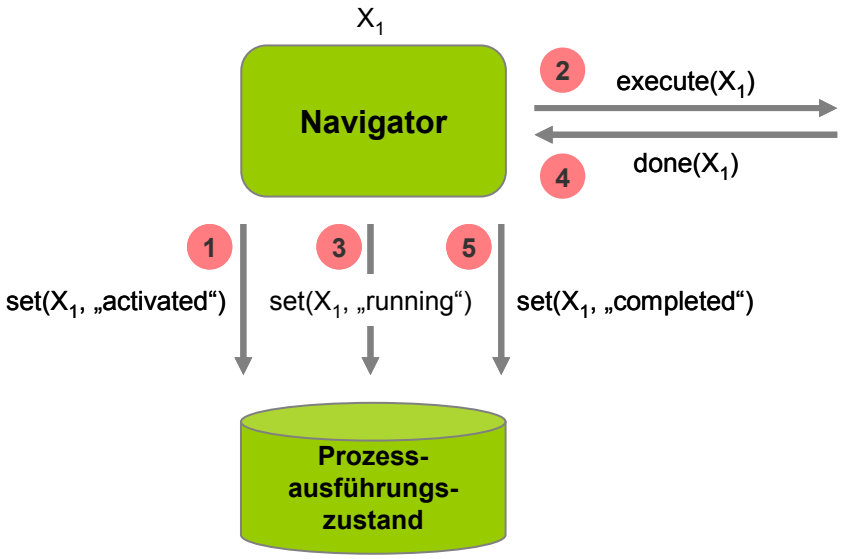


Abbildung 24: Prozessausführungszustand

Die Verwaltung des Prozessausführungszustandes wir in Abbildung 24 an einem Beispiel demonstriert. Sobald der Kontrollfluss die Aktivität X_1 erreicht hat, wird ihr Zustand auf „activated“ gesetzt (1). Dann wird die Ausführung der Aktivität gestartet (2) und der Zustand auf „running“ geändert (3). Sobald die Ausführung der Aktivität X_1 beendet ist (4) wird ihr Zustand auf „completed“ gesetzt (5).

Welche Zustände eine Aktivität, ein Prozess oder auch andere Komponenten einer Prozessinstanz durchlaufen können, hängt vom verwendeten Zustandsmodell ab. In [Ste10] wurde beispielsweise ein Zustandsmodell für WS-BPEL 2.0 definiert. Dieses Modell definiert zum einen die Zustände selbst und zum anderen die Zustandsübergänge, also die genauen Bedingungen, wann sich ein Zustand ändert. Ein weiteres Zustandsmodell für die Ausführung von Prozessmodellen wird von der Workflow Management Coalition definiert [Hol95].

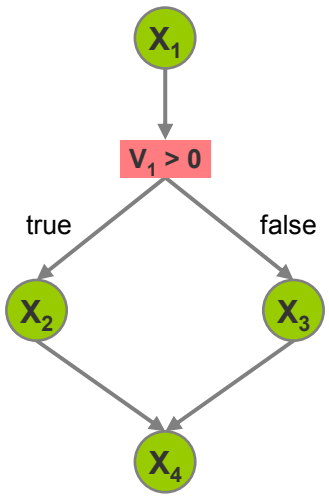
Prozessdaten und Prozessausführungszustand bilden gemeinsam eine vollständige Beschreibung einer konkreten Prozessinstanz. Der Navigator benötigt direkt oder indirekt Zugriff auf diese Daten. Die möglichen Aktionen des Navigators werden in jedem Schritt vom aktuellen Prozessausführungszustand sowie dem zugrundeliegenden Prozessmodell bestimmt. Die Prozessdaten können ebenfalls vom Navigator benötigt werden, wenn etwa Bedingungen, die auf diesen Daten basieren, ausgewertet werden müssen (siehe auch Kapitel 5.3.4).

Der Navigator benötigt also zum einen die Möglichkeit, den Prozessausführungszustand einer Prozessinstanz zu speichern und diesen Zustand anhängig von der Ausführung zu verändern. Zum anderen gilt dies auch für die Prozessdaten. Sie werden vom Navigator erzeugt, gelesen und geschrieben. Die Manipulation der Daten selbst geschieht in den meisten Fällen durch die Aktivitäten des Prozesses.

5.3.4. Auswertung von Ausdrücken

Die Navigation durch einen Prozess ist nicht nur vom Prozessmodell, sondern auch von den Prozessdaten abhängig. Kontrollfluss kann in den meisten Modellierungssprachen auch bedingt formuliert werden. Abbildung 25 zeigt ein Beispiel für bedingten Kontrollfluss. Auf der linken Seite ist ein einfaches Prozessmodell in Form eines Graphen dargestellt. Die Knoten entsprechen Aktivitäten, die Kanten repräsentieren den Kontrollfluss. Auf der rechten Seite der Abbildung ist das gleiche Prozessmodell mit Hilfe von Pseudocode beschrieben.

Nachdem die Aktivität X_1 ausgeführt wurde, wird entweder die Aktivität X_2 oder die Aktivität X_3 ausgeführt. Anschließend wird dann in beiden Fällen die Aktivität X_4 ausgeführt. Die Entscheidung, welche der beiden Aktivitäten X_2 und X_3 ausgeführt wird, hängt von der Bedingung „$V_1 > 0$“ ab. Wird diese Bedingung als „wahr“ („true“) ausgewertet, dann wird X_2 aktiviert. Wird die Bedingung dagegen als „falsch“ („false“) ausgewertet, wird X_3 aktiviert.




```
X1;
if (V1 > 0) then
  X2;
else
  X3;
endif
X4;
```

Abbildung 25: Bedingter Kontrollfluss

Eine (boolesche) Bedingung ist immer auch ein Ausdruck, wie von allgemeinen Programmiersprachen bekannt. Im Folgenden wird von der Auswertung von Ausdrücken gesprochen. Dies ist jedoch gleichzusetzen mit der Auswertung von Bedingungen, da Bedingungen immer mit Hilfe von Ausdrücken formuliert werden.

Um Ausdrücke zur Laufzeit auswerten zu können, ist in den meisten Fällen Zugriff auf die entsprechenden Variablen nötig. Um den Ausdruck „$V_1 > 0$“ auswerten zu können, wird der aktuelle Wert der Variablen V_1 benötigt. Abbildung 26 stellt dar, wie die Auswertung eines Ausdrucks zur Laufzeit erfolgen kann. Der Navigator bestimmt zunächst die aktuellen Werte aller zur Auswertung des Ausdrucks benötigten Variablen. Anschließend wird der Ausdruck sowie die dafür benötigten Variablen an eine dafür geeignete Komponente übergeben. Diese Komponente wertet den Ausdruck aus und gibt das Ergebnis („wahr“ oder „falsch“) an den Navigator zurück.

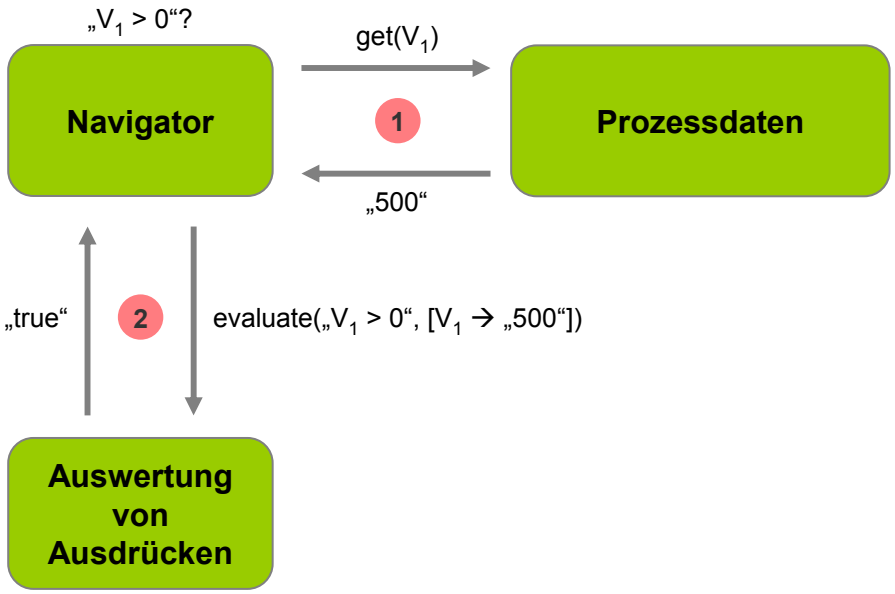


Abbildung 26: Auswertung von Ausdrücken, Szenario 1

Die Formulierung von Ausdrücken hängt vom zugrundeliegenden Datenmodell ab. Handelt es sich bei den Variablen beispielsweise um Java Objekt, dann werden Ausdrücke in Java Notation formuliert. Liegen die Variabeln als XML Elemente vor, dann können Bedingungen in der Form von XPath Ausdrücken angegeben werden. Beachtet man dieses, dann hat die in Abbildung 26 dargestellte Vorgehensweise einige Nachteile. Der Prozessdatenspeicher bestimmt das zugrundeliegende Datenmodell. Der Navigator muss die zu diesem Datenmodell passende Ausdruckssprache verstehen, da er anhand des Ausdrucks die zu dessen Auswertung benötigten Variablen bestimmen muss. Auch die Auswertungskomponente muss die zum Datenmodell des Prozessdatenspeichers passende Ausdruckssprache kennen, da sie für die eigentliche Auswertung der Ausdrücke zuständig ist. Alle beteiligten Komponenten sind also abhängig vom Prozessdatenspeicher, da er das Datenmodell und damit auch die dazugehörige Ausdruckssprache bestimmt.

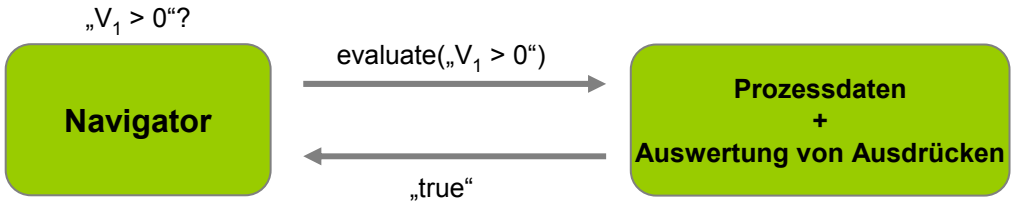


Abbildung 27: Auswertung von Ausdrücken, Szenario 2

Um die einzelnen Komponenten einer Workflow Engine möglichst unabhängig voneinander aufzubauen, bietet es sich an, die Funktionalität zur Auswertung von Ausdrücken in den Prozessdatenspeicher zu integrieren bzw. mit diesem in eine einzige Komponente zusammen zu fassen. Ein solcher Aufbau und die daraus resultierende Vorgehensweise sind in Abbildung 27 dargestellt. Der Navigator übergibt den auszuwertenden Ausdruck direkt an den Prozessdatenspeicher. Dieser bestimmt intern die Werte der benötigten Variablen, wertet den Ausdruck aus, und gibt das Ergebnis an den Navigator zurück. In diesem Szenario muss der Navigator die Ausdruckssprache selbst nicht verstehen. Dadurch ist er vom Datenmodell des Prozessdatenspeichers unabhängiger.

5.3.5. Programmaufrufkomponente

Prozessmodelle spezifizieren sowohl Kontroll- als auch Datenfluss im Bezug auf Aktivitäten. Diese Aktivitäten repräsentieren bestimmte Aktionen, welche während der Prozessausführung durch eine dazugehörige Implementierung ausgeführt werden. Der Begriff der Implementierung ist hier zunächst ganz allgemein im Sinne von Ausführen oder durchführen zu verstehen. Eine Aktivität wird nicht zwingend durch ein Programm implementiert, sie kann beispielsweise auch durch Menschen implementiert werden.

Wird eine Aktivität während der Ausführung einer Prozessinstanz aktiviert, so bedeutet dies, dass die dazugehörige Implementierung aufgerufen werden muss. Dieser Implementierung können Daten übergeben werden und sie kann bei ihrer Beendigung Daten zurückgeben.

Im Folgenden werden zwei Alternativen für die Umsetzung von Aktivitäten betrachtet. Diese Unterteilung wird beispielsweise auch vom Reference Model der Workflow Management Coalition reflektiert (Kapitel 5.1.1). Die erste Alternative basiert auf dem Konzept der drei Dimensionen eines Workflows und dem Begriff des Workitems. Die zweite Alternative betrachtet die Ausführung einer Aktivität als Aufruf eines Programms.

Ausführung von Aktivitäten mit Workitems

In Kapitel 1.2.1 wurde bereits das Konzept der drei Dimensionen eines Workflows nach [LR00] vorgestellt. Die erste Dimension ist die Prozesslogik (der Kontrollfluss). Sie bestimmt, wann welche Aktivität ausgeführt wird und gibt ebenfalls den Datenfluss an. Basierend auf der zweiten Dimension, der Organisationsstruktur, wird angegeben, wer eine bestimmte Aktivität ausführen soll. Diese Ausführenden bezeichnet man auch als Agenten. Die IT Infrastruktur bildet die dritte Dimension, hier wird angegeben, welche IT Ressourcen zur Ausführung einer Aktivität benötigt werden.

Basierend auf diesen drei Dimensionen wird eine Aktivität als ein 3-Tupel beschrieben. Das Tupel gibt an, welche Aktivität von welchen Agenten und unter Zuhilfenahmen welcher IT Ressourcen durchgeführt werden soll. Die Angabe der Agenten erfolgt in Form einer sogenannten *Staff Query*. Eine Staff Query ist eine Abfrage auf der Organisationsdatenbank, sie entkoppelt die Modellierung und die Ausführung eines Prozessmodells. Zur Modellierungszeit müssen die konkreten Agenten nicht bekannt sein, es wird allgemein formuliert, wer eine Aktivität ausführen soll (z.B.: „alle Agenten mit Level > 3 und Sicherheitsstufe > 5“).

Kann während der Prozessausführung eine Aktivität ausgeführt werden, wird zunächst die mit dieser Aktivität verknüpfte Staff Query ausgeführt. Das Ergebnis dieser Abfrage ist eine Menge von konkreten Agenten. Für jeden Agenten wird dann ein sogenanntes Workitem erstellt. Ein Workitem ist wiederum ein 3-Tupel, bestehend aus dem Agenten, der Aktivität sowie der zugeordneten IT Ressource. Die so erzeugten Workitems werden anschließend in der Workitem Datenbank abgelegt. Die Agenten selbst greifen auf die Workitem Datenbank zu, fragen die für sie bestimmten Workitems ab und führen dann die in den Workitems beschriebenen Aktivitäten unter Zuhilfenahmen der angegebenen Ressourcen aus.

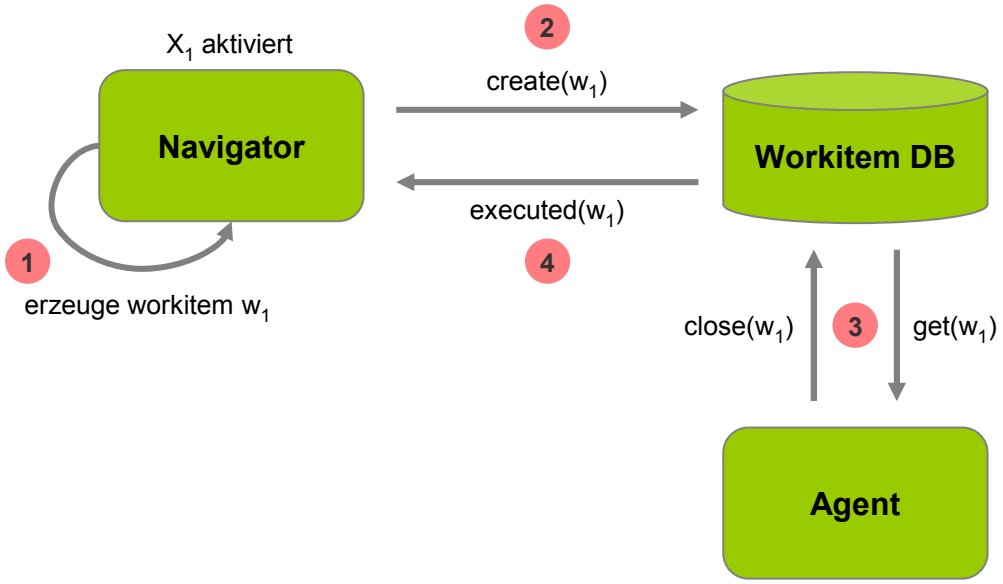


Abbildung 28: Ausführung einer Aktivität mit Workitems

Die prinzipielle Funktionsweise der Ausführung einer Aktivität mit Workitems ist in Abbildung 28 dargestellt. Der Navigator hat die Aktivität X_1 aktiviert, muss sie also ausführen. Als erstes werden ein oder mehrere entsprechende Workitems erzeugt (1) und anschließend in der Workitem Datenbank abgelegt (2). Die Agenten greifen auf die Workitem Datenbank zu, um die für sie relevanten Workitems zu bestimmen. Liegt ein passendes Workitem vor, so lesen sie es aus der Datenbank, führen die eigentliche Aktivität aus, und markieren das Workitem abschließend als abgearbeitet (3). Ist das Workitem in der Workitem Datenbank als abgearbeitet markiert, kann der Navigator die Prozessausführung bei X_1 fortsetzen (4).

Aus Sicht der Workflow Engine und auch aus Sicht des Navigators bedeutet die Ausführung einer Aktivität also lediglich das Erzeugen und die Zurverfügungstellung der entsprechenden Workitems. Die Ausführung selbst wird von Agenten übernommen. Das Konzept der Workitems wird vor allem im Zusammenhang mit Aktivitäten, die von Menschen ausgeführt werden, verwendet. Ein Workitem definiert eine durch den Menschen zu erledigende Aufgabe. Alle einer Person zugeordneten Workitems werden dann von einer Worklist verwaltet, einer Liste aller von dieser Person zu erledigenden Aufgaben.

Ausführung von Aktivitäten durch Programmaufruf

Das Konzept der Workitems bietet eine sehr allgemeine Sicht auf Aktivitäten und ihre möglichen Implementierungen. Schränkt man die Sicht auf Aktivitäten jedoch derart ein, dass man sie lediglich als Programmaufrufe versteht, dann ändert sich auch das Vorgehen zur Ausführung einer Aktivität. Die Aktivität selbst enthält alle relevanten Parameter, die zur Ausführung des mit ihr assoziierten Programms notwendig sind.

Im einfachsten Fall kann der Navigator also, sobald er feststellt, dass eine Aktivität ausgeführt werden kann, das entsprechende Programm direkt aufrufen. Dieses Vorgehen beinhaltet jedoch mehrere Probleme. Handelt es sich um einen synchronen und lange andauernden Programmaufruf, dann ist der Navigator die gesamt Zeit über blockiert. Er kann die Ausführung der aktuellen oder auch anderer Prozessinstanzen nicht weiter vorantreiben. Ein weiterer Nachteil resultiert daraus, dass es viele unterschiedliche Paradigmen und Techniken gibt, Programme aufzurufen. Der Navigator muss, wenn er die Programme selbst aufruft, alle diese Paradigmen und Techniken unterstützen.

Der Aufruf von Programmen kann aber auch in eine separate Komponente ausgelagert werden. Der Navigator übergibt dieser Komponente alle Informationen, welche diese zum Aufruf des entsprechenden Programms benötigt. Die Aufrufkomponente muss dann in der Lage sein, alle geforderten Aufrufparadigmen und –techniken zu unterstützen. Der Navigator wird damit vom eigentlichen Aufruf von Programmen entkoppelt. Er muss diese Aufrufe nicht selbst ausführen, er delegiert sie an die Aufrufkomponente.

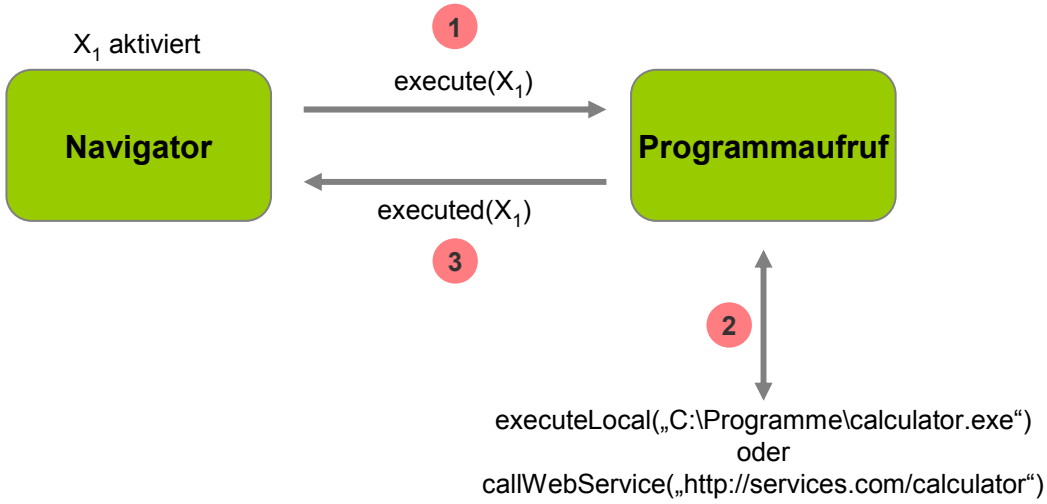


Abbildung 29: Ausführung einer Aktivität durch Programmaufruf

Die Funktionsweise der Programmaufrufkomponente ist in Abbildung 29 an einem Beispiel dargestellt. Der Navigator hat die Aktivität X_1 aktiviert, er muss sie also ausführen. Die Aktivität spezifiziert den Programmaufruf, den sie repräsentiert, genauer. Diese Spezifikation übergibt der Navigator an die Komponente Programmaufruf (1). Diese Komponente ist in der Lage, den durch die Aktivität X_1 definierten Programmaufruf durchzuführen (2). Sie muss prinzipiell in der Lage sein, unterschiedlichste Programmaufrufparadigmen und -techniken zu interpretieren und zu implementieren. Die Aktivität X_1 könnte, wie im Beispiel dargestellt, entweder einen lokalen Programmaufruf, alternativ aber beispielsweise auch einen Web Service Aufruf definieren. Ist der Programmaufruf abgearbeitet, bekommt der Navigator von der Programmaufrufkomponente eine entsprechende Rückmeldung (3). Er kann die Prozessausführung bei der Aktivität X_1 nun fortsetzen.

Eine weitere Möglichkeit, die Workflow Engine noch mehr von der eigentlichen Implementierung einer Aktivität zu entkoppeln, ist die Nutzung von Web Service Technologie. Diese Variante wird beispielsweise von WS-BPEL, Workflow Grammatiken oder teilweise auch von YAWL genutzt. Web Services bieten nach außen eine durch die *Web Service Description Language (WSDL)* [WSDL07] definierte Schnittstelle an. Diese Schnittstellenbeschreibung ist unabhängig von der eigentlichen Schnittstelle des dahinterliegenden Programms. Die Bereitstellung von Programmen über Web Service Schnittstellen bietet dem Nutzer dieser Programme damit eine einheitliche Sicht auf ihre Schnittstellen an. Die eigentliche Implementierung kann wiederum unterschiedliche Aufrufparadigmen und -techniken benötigen. Dies wird durch die Web Service Schnittstelle jedoch verborgen.

Aus Sicht der Workflow Engine wird durch die Nutzung von Web Services der Programmaufruf vereinfacht, sie muss lediglich die Fähigkeit zu Web Service Ausrufen implementieren. Die Abbildung eines Web Service Aufrufs auf einen Aufruf des dahinterliegenden Programms muss durch den Anbieter der Web Service Schnittstelle implementiert werden.

5.3.6. Erzeugen von Prozessinstanzen, Übergabeparameter

In den bisher dargestellten Beispielen für die Funktionsweise des Navigators wurde immer davon ausgegangen, dass eine konkrete Prozessinstanz bereits ausgeführt wird. Jede Prozessinstanz muss jedoch zu Beginn ihrer Ausführung einmalig erzeugt und gestartet werden. Der Start einer Prozessinstanz wird, allgemein formuliert, von außen signalisiert. Wie genau diese geschieht, kann sehr unterschiedlich sein. In WS-BPEL geschieht die Instanziierung von Prozessinstanzen durch den Empfang bestimmter Nachrichten. Es ist aber ebenso denkbar, dass Prozessinstanzen über einen Funktionsaufruf oder die Auslösung eines Events erzeugt und gestartet werden. Die Implementierung und Integration der entsprechenden Mechanismen ist Teil des Workflow Management Systems. Wesentlich für den Navigator ist es, dass er eine Schnittstelle nach außen bietet, über welche die Erzeugung einer neuen Prozessinstanz und ihre Ausführung gestartet werden kann.

Abhängig vom Prozessmodell können Prozessinstanzen zu Beginn ihrer Ausführung Daten übergeben bekommen und zum Ende ihrer Ausführung Ergebnisdaten zurückgeben. Der Navigator muss in diesem Fall entsprechende Schnittstellen implementieren.

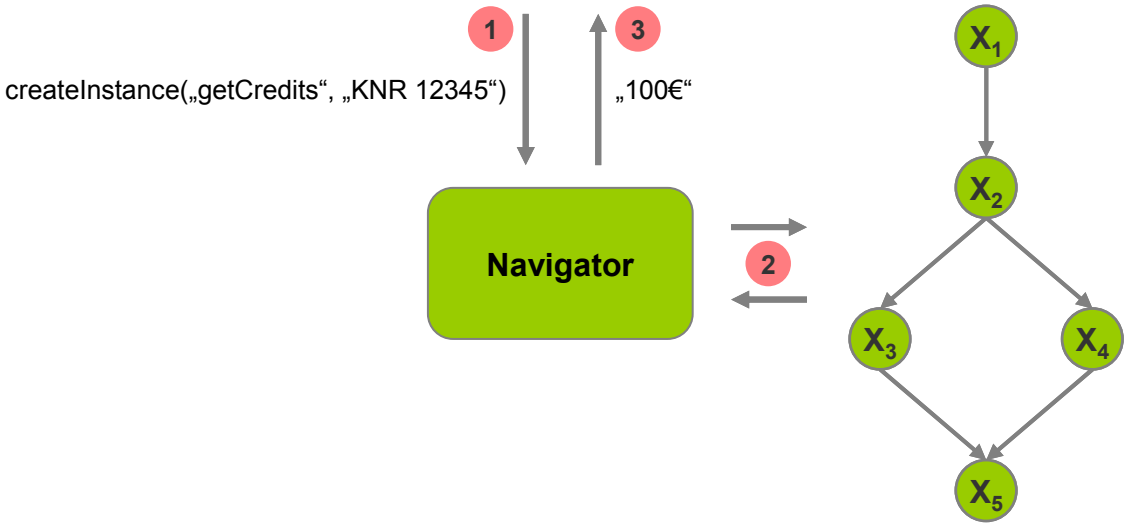


Abbildung 30: Aktivierung des Navigators

Die Aktivierung des Navigators mit Eingabe- und Ausgabedaten ist in Abbildung 30 an einem Beispiel dargestellt. Die Erzeugung einer neuen Prozessinstanz wird von außen initialisiert. Dem Navigator wird das zu instanziierende Prozessmodell „getCredits“ sowie die von diesem Prozessmodell benötigten Eingabedaten „KNR 12345“ übergeben (1). Anschließend führt der Navigator diese Prozessinstanz aus (2), er navigiert durch das Prozessmodell, ruft Aktivitäten auf, wertet Ausdrücke aus und liest sowie schreibt Daten. Ist die Ausführung der Prozessinstanz beendet, gibt der Navigator die durch das Prozessmodell spezifizierten Ausgabedaten „100€“ nach außen zurück (3).

5.4. Ausführung von Workflows in einer Cloud Umgebung

Die Grundlagen zum Thema Cloud Computing wurden bereits in Kapitel 1.2.5 beschrieben. Betrachtet man eine Cloud auf der Ebene Platform as a Service (PaaS), dann kann die Cloud dem Nutzer eine Umgebung anbieten, in der Programme installiert und anschließend verwendet werden können. Eine solche PaaS Cloud kann, etwas verkürzt dargestellt, mit einem Application Server verglichen werden. Dieser spezielle Application Server wird vom Cloud Provider betrieben und gewartet, der Nutzer verwendet ihn lediglich nach Bedarf. Anbieter von PaaS Lösungen sind beispielsweise Google mit der Google AppEngine[11] oder Microsoft mit der Microsoft Windows Azure Cloud[12].

Die Modellierung von Prozessen, welche anschließend auf einer Workflow Engine ausgeführt werden, bezeichnet man auch als „*Programmieren im Grossen*“ [EMW+05]. Die Implementierung der einzelnen Funktionalitäten, welche durch das Prozessmodell orchestriert werden, wird dementsprechend als „*Programmieren im Kleinen*“ bezeichnet. Für den Bereich des „Programmieren im Kleinen“ existieren, wie bereits erwähnt, verschiedene PaaS Lösungen. Der entsprechende Schritt im Bezug auf „Programmieren im Grossen“ wäre es nun, eine PaaS Lösung anzubieten, auf welcher Prozessmodelle installiert und anschließend ausgeführt werden können.

In [Hoe10] wird beschrieben, welche Anforderungen an ein Workflow Management System gestellt werden, falls dieses in eine Cloud integriert und als PaaS Produkt angeboten wird. Die dort beschriebene Lösung wird als *Orchestration as a Service (OaaS)* bezeichnet. Die Architektur besteht kurzgefasst darin, bestehende Workflow Management Systeme mit zusätzlichen Diensten und Funktionen zu umgeben, um sie so „Cloud fähig“ zu machen. Dieser Aufbau ist in Abbildung 31 schematisch dargestellt.

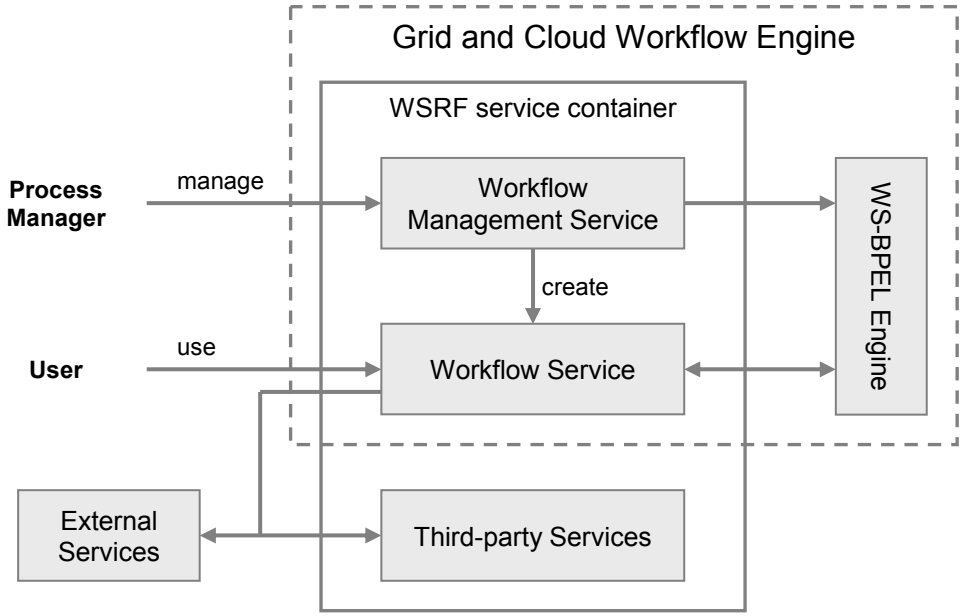


Abbildung 31: Komponenten einer Cloud Workflow Engine nach [Hoe10]

[11] http://code.google.com/appengine/

[12] http://www.microsoft.com/windowsazure/

Der in [Hoe10] entwickelte Ansatz ergibt sich aus der allgemeinen Beobachtung, dass die von einer Cloud angebotenen Funktionalitäten im Wesentlichen bereits vorhanden sind. Grundfunktionalitäten wie beispielsweise Application Server, Messaging System, relationale Datenbank oder eben auch Workflow Management System werden durch zusätzliche Funktionalitäten und nichtfunktionale Eigenschaften wie etwa einfache Benutzungsschnittstellen, Mehrbenutzerfähigkeit, Skalierbarkeit oder Abrechnungsmöglichkeiten angereichert. In [Hoe10] wird die Workflow Engine selbst als eine geschlossene Komponente betrachtet.

In dieser Arbeit soll dagegen eine Möglichkeit vorgestellt werden, wie das Wissen über die in Kapitel 5.3 vorgestellte Unterteilung einer Workflow Engine in bestimmte Komponenten bei der Entwicklung einer Cloud Workflow Engine hilfreich sein kann. Im Folgenden werden die Grundidee und zwei unterschiedliche Anwendungsszenarien vorgestellt.

5.4.1. Komponenten einer Workflow Engine in einer Cloud Umgebung

Die in dieser Arbeit vorgestellte Grundidee besteht darin, die einzelnen Komponenten einer Workflow Engine durch die Nutzung von Cloud Diensten zu implementieren. Der Navigator ist in einer klassischen Workflow Engine der Kern einer modularen, aber dennoch zentralen Anwendung, wie sie in Abbildung 32 dargestellt ist. Alle benötigten Komponenten sind Teil der Workflow Engine.

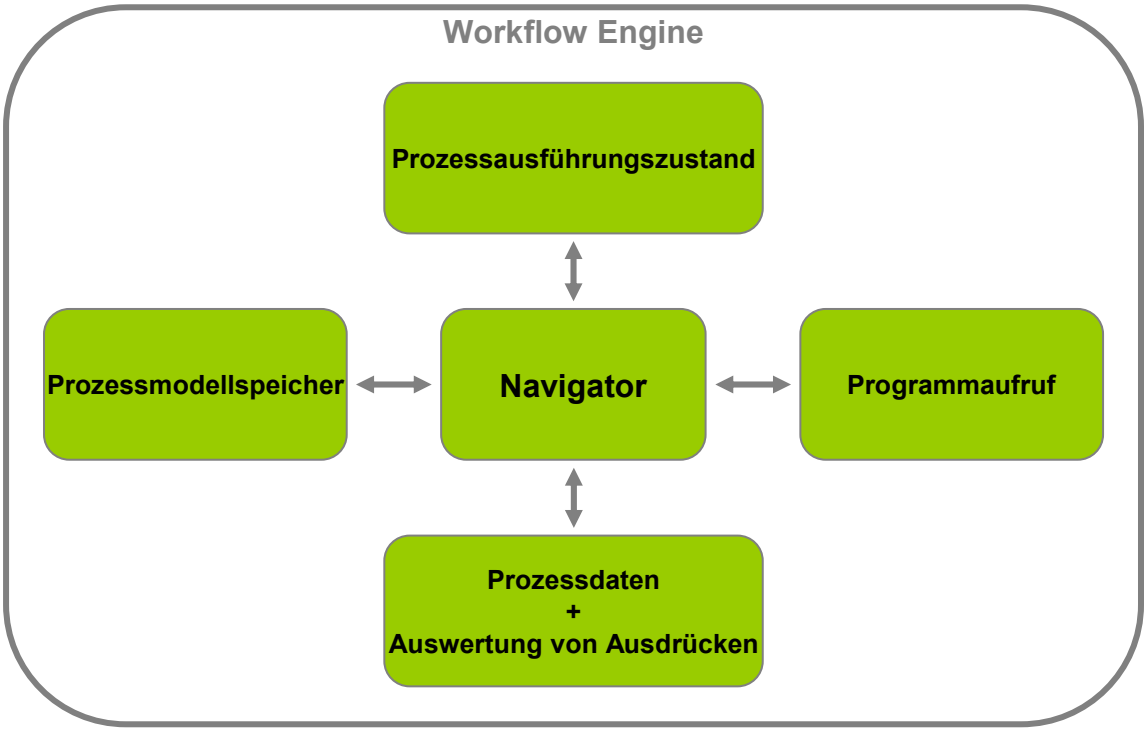


Abbildung 32: Workflow Engine als modulares System

Der Navigator einer Cloud Workflow Engine kann für die Umsetzung seiner Funktionalität aber auch verschiedene Dienste der Cloud nutzen. Die Workflow Engine ist damit weiterhin ein modular aufgebautes, gleichzeitig aber auch ein verteiltes System. Abbildung 33 zeigt die sich daraus ergebende mögliche Architektur einer Workflow Engine innerhalb einer Cloud Umgebung. Der Navigator bildet den Kern der Workflow Engine. Alle weiteren Funktionalitäten liegen außerhalb der eigentlichen Workflow Engine. Der Navigator nutzt unterschiedliche Dienste über entsprechende Schnittstellen, um alle weiteren Funktionalitäten zu implementieren.

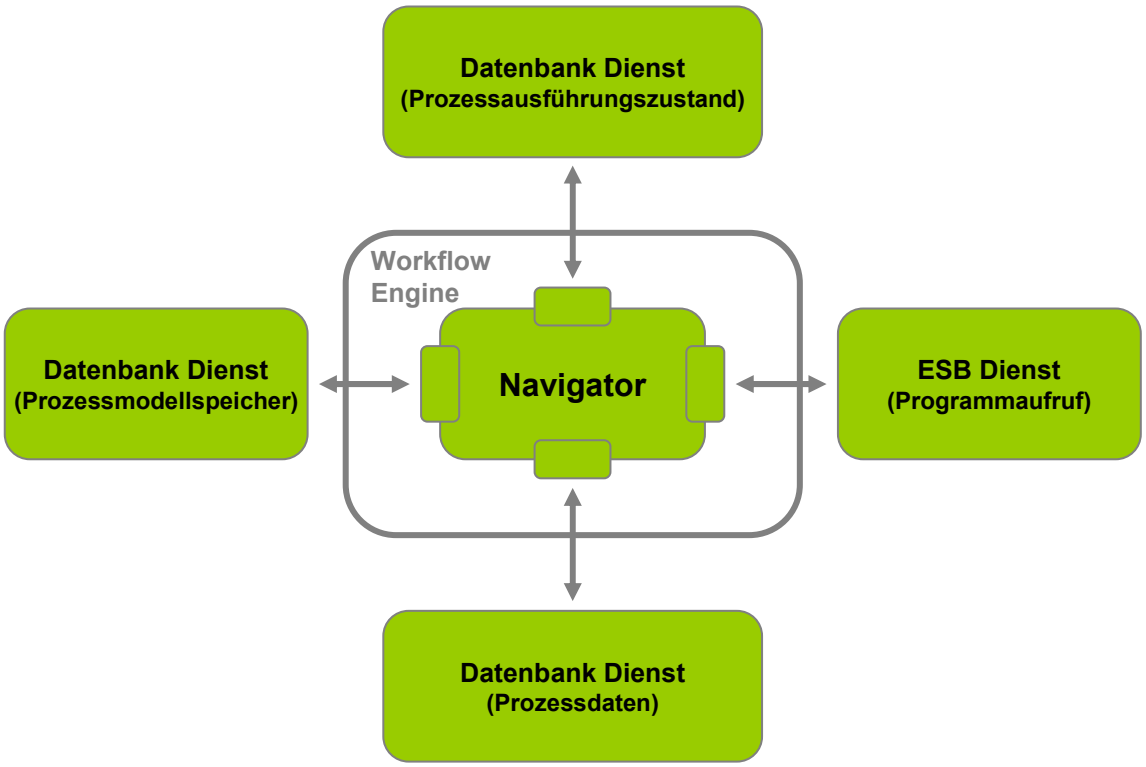


Abbildung 33: Navigator als Workflow Engine + Dienste

Im Folgenden wird beispielhaft gezeigt, wie die Aufteilung einer Workflow Engine und die Ausgliederung einzelner Komponenten in die Cloud dazu genutzt werden kann, den Aufbau einer Workflow Engine an ein bestimmtes Anwendungsszenario anzupassen.

5.4.2. Anwendungsszenario 1

Das erste Szenario betrachtet die Ausführung von *Production Workflows* auf einer Workflow Engine. Als Production Workflow werden laut [LR00] Prozessmodellen klassifiziert, die einen hohen geschäftlichen Wert besitzen und gleichzeitig oft wiederholt ausgeführt werden. Ist nun eine Workflow Engine ausgelastet, das heißt die maximale Anzahl von gleichzeitig ausführbaren Prozessinstanzen ist erreicht, dann muss eine weitere Workflow Engine installiert werden (der Prozess besitzt einen hohen geschäftlichen Wert, er muss auf jeden Fall ausgeführt werden). Verfügbare Prozessmodelle können dann, je nach Auslastung, auf jeder der beiden Wokflow Engines instanziiert werden. Eine Prozessinstanz wird jedoch immer nur auf genau einer Workflow Engine ausgeführt.

Abbildung 34 zeigt dieses Szenario mit zwei Workflow Engines, die gemeinsam innerhalb einer Cloud installiert sind. Beide Workflow Engines teilen sich sowohl die Komponente Prozessmodellspeicher als auch die Komponente Programmaufruf. Beide Workflow Engines benötigen Zugriff auf alle Prozessmodelle. Ebenfalls von beiden Workflow Engines benötigt wird die Möglichkeit zum Programmaufruf. Da eine Prozessinstanz aber immer nur von genau einer Workflow Engine ausgeführt wird, besteht keine Notwendigkeit, die Komponenten „Prozessdaten“ und „Prozessausführungszustand“ zwischen beiden Workflow Engines zu teilen. Aus diesem Grund besitzt in diesem Szenario jede Workflow Engine ihre eigenen entsprechenden Komponenten.

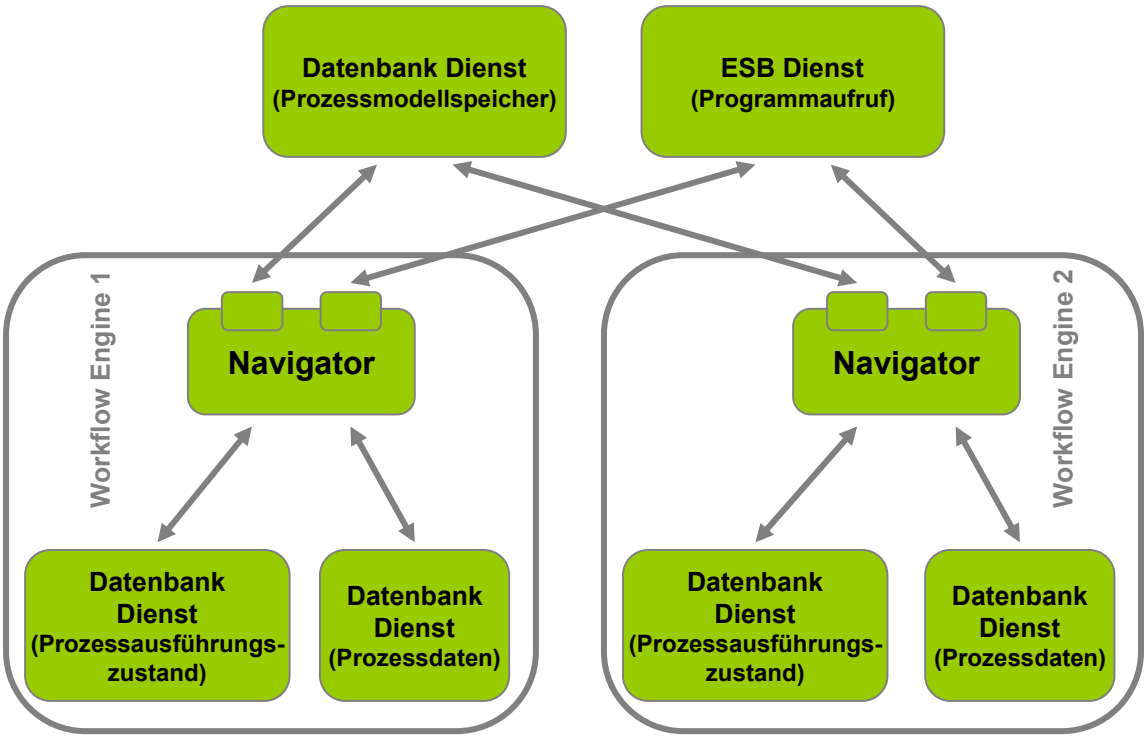


Abbildung 34: Anwendungsszenario 1

Der hier beschriebene Anwendungsfall nutzt also die Modularität der Workflow Engine aus, um gemeinsam benötigte Funktionalität in die Cloud auszulagern. Dies betrifft aber nur einen Teil der Komponenten, die restlichen Komponenten bleiben dem jeweiligen Navigator direkt zugeordnet.

5.4.3. Anwendungsszenario 2

Das zweite hier vorgestellt Anwendungsszenario betrachtet die Ausführung eines *Simulations-Workflows* (Scientific Workflow). Simulations- Workflows unterscheiden sich wesentlich von *Geschäfts- Workflows* (Business Workflows). Sie sind in den meisten Fällen nicht kontroll- sondern datengetrieben, sie werden nur wenige Male instanziiert und haben im Schnitt eine lange Laufzeit [TDG+07].

In [Prz11] wurde eine Methode vorgestellt, wissenschaftliche Workflows innerhalb einer Cloud Umgebung instanzbasiert zu installieren und dynamisch verteilt auszuführen. Beim Erzeugen einer Prozessinstanz wird das Prozessmodell zunächst in Fragmente unterteilt, parallele Ausführungsstränge werden beispielsweise in einzelne Fragmente unterteilt. Abbildung 35 zeigt ein Beispiel eines bereits in vier Fragmente unterteilten Prozessmodells. Dieses Prozessmodell soll in der dargestellten Cloud Umgebung installiert und ausgeführt werden. Zur Verfügung stehen dabei verschiedene virtuelle Maschinen (VM).

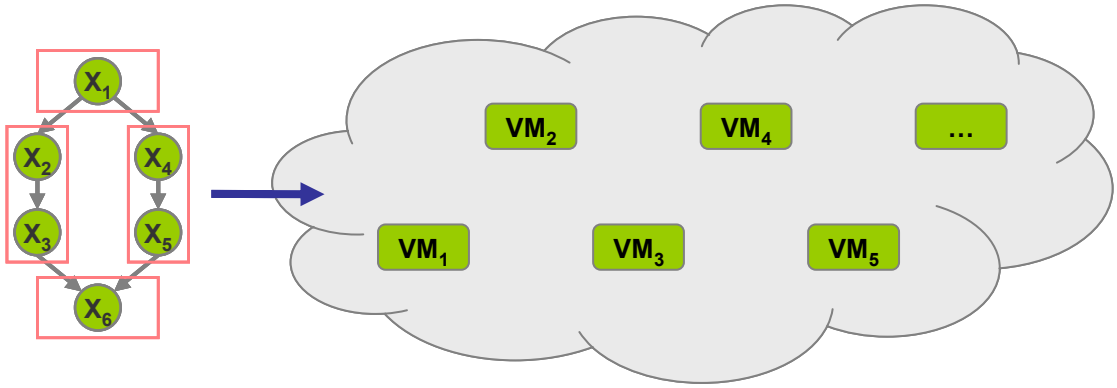


Abbildung 35: Ausgangssituation

Anschließend an die Fragmentierung des Prozessmodells werden die ersten Fragmente in der Cloud installiert. Die Cloud Umgebung wird dabei als IaaS (Infrastructure as a Service) Umgebung genutzt, sie bietet virtuelle Maschinen, aber keinerlei Softwarefunktionalität an. Es wird also nicht nur das Prozessfragment, sondern auch die dazugehörige Workflow Engine auf den virtuellen Maschinen installiert. Zu Beginn der Ausführung einer Prozessinstanz werden noch nicht alle Fragmente installiert, sondern nur die, bezogen auf die Ausführungsreihenfolge, ersten Fragmente. Die fehlenden Fragmente werden stückweise nachinstalliert, sobald die ersten der laufenden Fragmente beendet wurden.

Abbildung 36 zeigt die Installation und Ausführung eines Prozessmodells auf Basis der in Abbildung 35 vorgestellten Ausgangssituation. Für die Ausführung eines Prozessfragments wird je eine virtuelle Maschine (VM) benötigt. Auf dieser werden zunächst eine Workflow Engine (WE) und anschließend das auszuführende Prozessfragment installiert. Die Fragmente werden dann schrittweise ausgeführt. Nach Beenden des ersten Fragmentes werden beide Folgefragmente gestartet (1). Nachdem diese beendet wurden, wird das letzte Fragment aktiviert und ausgeführt (2).

Die Fragmentierung eines wissenschaftlichen Workflows erlaubt es, einzelne Fragmente des Prozesses für eine optimale Ausführung auf unterschiedliche virtuelle Maschinen der Cloud Umgebung zu verteilen. Durch das instanzbasierte Deployment werden die einzelnen Fragmente erst kurz vor ihrer Aktivierung installiert, sie können so immer optimal platziert werden. Ein vollständiges Deployment zum Instaziierungszeitpunkt würde eine mögliche Dynamik der Cloud Umgebung außer Acht lassen. Die Verteilung der Fragmente wäre in diesem Fall zur Instanziierungszeit optimal, aber nicht unbedingt auch zur Laufzeit der einzelnen Fragmente.

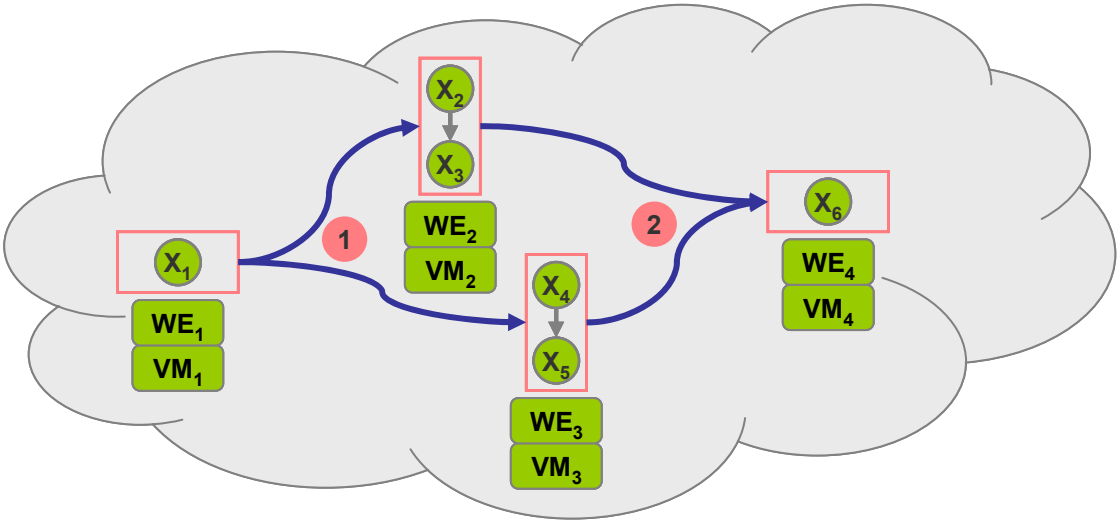


Abbildung 36: Installation und Ausführung

Das in [Prz11] vorgestellte Verfahren zum dynamischen instanzbasierten Deployment einzelner Prozessfragmente installiert auf den virtuellen Maschinen der Cloud Umgebung jeweils eine vollständige Workflow Engine. In Abbildung 37 ist dargestellt, wie dieses Verfahren unter Ausnutzung des modularen Aufbaus einer Workflow Engine angepasst werden könnte. Auf den einzelnen virtuellen Maschinen der Cloud Umgebung wird im Wesentlichen nur noch der Navigator installiert. Alle weiteren Komponenten der Workflow Engine werden als Cloud Dienste genutzt. Dieser Aufbau reflektiert das vorgestellte Anwendungsszenario. Es wird nur eine Prozessinstanz ausgeführt, dementsprechend gibt es auch nur genau eine Komponente Prozessdaten und genau eine Komponente Ausführungszustand. Diese verwalten den Zustand der Prozessinstanz zentral, die Ausführung erfolgt jedoch verteilt durch die einzelnen Workflow Engines (Navigatoren).

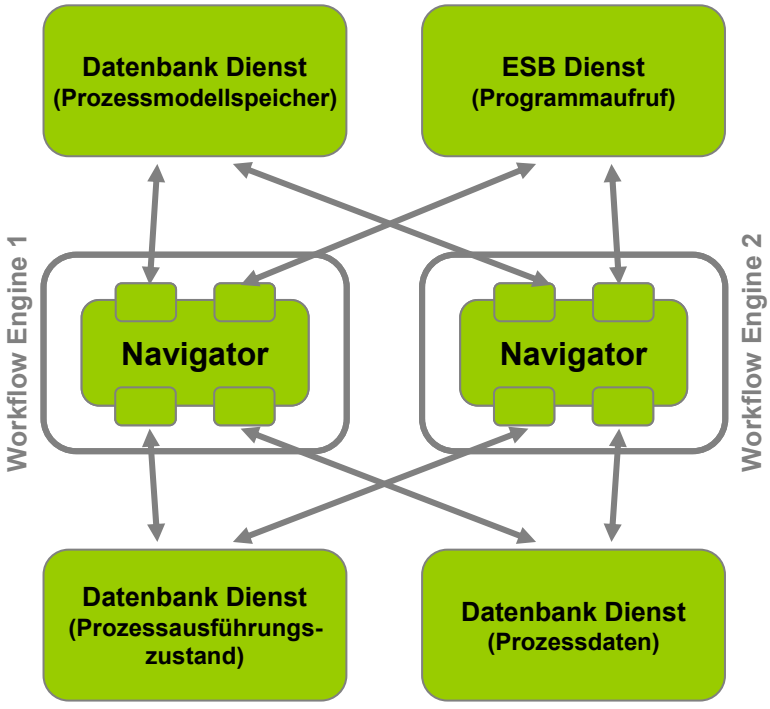


Abbildung 37: Anwendungsszenario 2

6 Entwurf einer Workflow Engine für die Ausführung von Workflow Grammatiken

In diesem Kapitel wird der Entwurf einer Workflow Engine für die Ausführung von Workflow Grammatiken beschrieben. Workflow Grammatiken, wie sie in Kapitel 3 beschrieben wurden, werden durch die hier beschriebene Workflow Engine nicht direkt ausgeführt, sondern zunächst in eine interne Repräsentation überführt. Die einzelnen Aspekte dieser Transformation werden in Kapitel 6.1 beschrieben. Der modulare Aufbau und die Funktionsweise der einzelnen Komponenten einer Workflow Engine für die Ausführung von Workflow Grammatiken werden anschließend in Kapitel 6.2 vorgestellt.

6.1. Überführung von Workflow Grammatiken ein eine interne Repräsentation

In Kapitel 3 wurden das Konzept und der konkrete Aufbau von Workflow Grammatiken vorgestellt. Die Ausführung von Workflow Grammatiken, also das Erzeugen von Wörtern aus einer gegebenen Grammatik, wurde anschließend in Kapitel 4 näher untersucht. Mögliche Verfahren dafür wurden für alle Grammatiktypen der Chomsky Hierarchie vorgestellt. Ein wesentliches Ergebnis war, dass der Aufwand zum Erzeugen von Wörtern aus einer Grammatik abhängig von der Klasse der Grammatik zunimmt. Je größer die Klasse der Grammatik[13], desto mehr Aufwand benötig ein passendes Verfahren. Die in Kapitel 3 vorgestellten Workflow Grammatiken sind in ihrer dargestellten Form vom Typ 0. In [Vuk11] wurde jedoch gezeigt, dass sich die Regeln von Workflow Grammatiken immer so umbauen lassen, dass sie kontextsensitiv sind. Die Darstellung als Typ 0 Grammatiken dient lediglich der besseren Darstellung und Verständlichkeit. Im Folgenden wird in Kapitel 6.1.1 gezeigt, wie die Regeln einer Workflow Grammatik so umgebaut werden können, dass sie eine kontextfreie Grammatik bilden. Damit wird erreicht, dass die Ausführung von Workflow Grammatiken einfacher und effizienter organisiert werden kann.

Regeln von Workflow Grammatiken können Aktionen wie beispielsweise den Aufruf eines Web Services oder die Manipulation von Prozessdaten repräsentieren. Bei der Ausführung einer Workflow Grammatik müssen nicht nur ihre Produktionsregeln auf das jeweils aktuelle Teilwort der Ableitung angewendet werden, sondern es müssen vor allem auch die damit verbundenen Aktionen ausgeführt werden. In Kapitel 6.1.2 wird gezeigt, wie eine Workflow Engine die Regeln einer Grammatik organisiert und interpretiert, um die durch die Regeln beschriebenen Aktionen effizient ausführen zu können.

Workflow Grammatiken bestehen zum einen aus Prozessstrukturregeln und zum anderen aus generischen Regeln. Prozessstrukturregeln werden immer explizit modelliert, sie beschreiben die eigentliche Struktur eines Prozesses. Die generischen Regeln werden dagegen nicht explizit modelliert. Sie beschreiben allgemeine Eigenschaften eines Prozessmodells, die im Wesentlichen unabhängig von einem konkreten Prozessmodell sind. Es gibt beispielsweise generische Regeln, welche die Lebensdauer und die Zugreifbarkeit von Variablen modellieren. Eine Workflow Engine für die Ausführung von Workflow Grammatiken bekommt immer nur Prozessstrukturregeln übergeben. Die Ausführung der generischen Regeln muss also implizit durch die Workflow Engine implementiert werden. Im Folgenden wird in Kapitel 6.1.3 gezeigt, wie die in Workflow Grammatiken implizit enthaltenen generischen Regeln von einer Workflow Engine umgesetzt werden können.

[13] Die Klasse der Typ 2 Grammatiken ist beispielsweise größer als die Klasse der Typ 3 Grammatiken, da es echt mehr Typ 2 Grammatiken als Typ 3 Grammatiken gibt. Allgemein gilt also, dass die Klasse der Typ x Grammatiken größer ist als die Klasse der Typ x+1 Grammatiken.

Die Kapitel 6.1.1, 6.1.3 und 6.1.2 beschreiben gemeinsam, wie eine gegebene Workflow Grammatik transformiert und aufbereitet wird, so dass sie von einer Workflow Engine effizient ausführbar ist. Sie wird in eine an die Ausführung angepasste interne Repräsentation überführt. Alle in diesen Kapiteln beschriebenen Verfahren werden einmalig bei der Installation eines Prozessmodells angewendet.

6.1.1. Transformation in kontextfreie Grammatiken

Die Grundidee bei der Transformation einer Workflow Grammatik ein eine kontextfreie Grammatik besteht darin, bestimmte Nichtterminale aus den Regeln zu entfernen und im Gegenzug dazu die entsprechende Grammatikregeln mit Metadaten anzureichern. Durch den Umbau der Grammatik dürfen keine Informationen verloren gehen. Die umgewandelte Grammatik muss inhaltlich äquivalent zu der Ausgangsgrammatik sein. Sie muss das gleiche Prozessmodell beschreiben.

Im Folgenden wird betrachtet, welche Regeln einer Workflow Grammatik kontextsensitiv sind. Für jeden Fall wird beschrieben, wie solche Regeln kontextfrei gemacht werden können. Grundlegend gilt für alle folgenden Abschnitte, dass für die Kontextfreiheit nur die linke Seite einer Produktionsregel relevant ist.

Datenfluss / Variablen

Der Zugriff von Aktivitäten auf Daten bzw. Variablen wird in Workflow Grammatiken dadurch modelliert, dass die Aktivierungs- und Beendigungsregel der entsprechenden Aktivität die benötigten Variabeln auf beiden Seiten der Regel enthält. Alle Regeln, welchen Datenfluss beschreiben, sind damit kontextsensitiv.

Enthält eine Regel auf beiden Seiten das gleiche Variablen Nichtterminal, dann wird dieses Nichtterminal aus der Regel entfernt. Im Gegenzug dazu wird in den Metadaten dieser Regel vermerkt, dass sie den Zugriff auf diese Variable modelliert. Die Art des Datenzugriffs, also lesender oder schreibender Zugriff, ergibt sich aus dem Typ der Regel. Aktivierungsregeln von Aktivitäten beschreiben lesenden Datenzugriff, Beendigungsregeln von Aktivitäten modellieren schreibenden Datenzugriff.

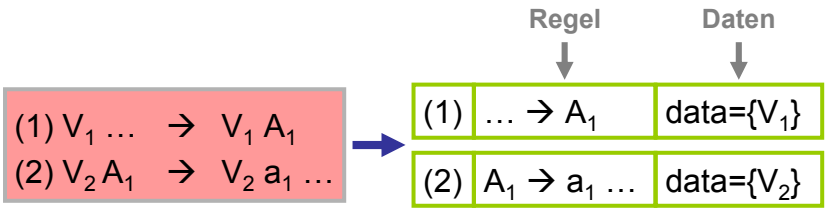


Abbildung 38: Variablenzugriff

Abbildung 38 zeigt an einem kleinen Beispiel, wie Variablenzugriffe aus den Regeln entfernt und in den Metadaten der Regel vermerkt werden. Die gezeigten Regeln werden dadurch kontextfrei.

Außer in Aktivierungs- und Beendigungsregeln von Aktivitäten können Variablen auch auf der linken Seite von Regeln vorkommen, in denen das Ende eines Scopes modelliert wird. In Kapitel 6.1.2 wird gezeigt, dass diese Regeln komplett aus der Grammatik entfernt werden. Damit muss für derartige Regeln keine Kontextfreiheit hergestellt werden, sie werden hier nicht betrachtet.

Beendigung von Scopes

Scopes beschreiben in Workflow Grammatiken den Lebensbereich von Variablen. Es gibt sowohl Regeln, in denen Scopes erzeugt werden, als auch Regeln, in denen Scopes beendet werden. Das Beenden eines Scopes bedeutet das Löschen seiner Grenzen sowie das Löschen aller in ihm

lebenden Variabeln. Diese Regeln sind immer kontextsensitiv, sie enthalten auf der linken Seite mindestens zwei Nichtterminale, welche die Grenzen des Scopes symbolisieren. Wie bereits im vorherigen Abschnitt erwähnt wurde, werden diese Regeln komplett aus der Grammatik entfernt (Kapitel 6.1.2). Aus diesem Grund werden Beendigungsregeln für Scopes hier nicht weiter betrachtet.

Join

Die Modellierung von Synchronisationspunkten (auch als Join bzw. Zusammenführung bezeichnet) in einem Prozessmodell erfolgt in Workflow Grammatiken durch kontextsensitive Regeln. Jeder der zu synchronisierenden Ausführungsstränge eines Prozessmodells erzeugt am Ende seiner Ausführung durch entsprechende Produktionsregeln ein Platzhalter- Nichtterminal. Die Synchronisation von x Ausführungssträngen wird dann durch eine Regel beschrieben, die auf der linken Seite genau x entsprechende Platzhalter enthält. Es gibt dabei immer genau einen festen und x-1 mobile Platzhalter.

Regeln dieser Art werden so umgebaut, dass sie auf der linken Seite nur noch genau einen Platzhalter enthalten. Die mobilen Platzhalter werden aus der linken Seite der Regel entfernt, lediglich der feste Platzhalter bleibt dann übrig. Die Information, wie viele der mobilen Platzhalter benötigt werden, um diese Regel auf ein Teilwort anwenden zu können, wird in den Metadaten der Regel hinterlegt. Vor der Ausführung einer solchen Regel muss dann zur Laufzeit überprüft werden, ob bereits eine passende Anzahl von entsprechenden Platzhaltern erzeugt wurde.

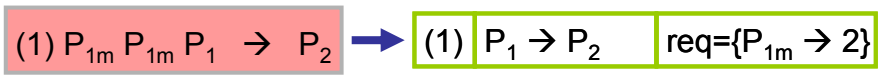


Abbildung 39: Mobile Platzhalter

Ein Beispiel für einen solchen Umbau einer Regel ist in Abbildung 39 dargestellt. Die mobilen Platzhalter P_{1m} werden aus der linken Seite der Regel entfernt, ihre Anzahl wird in den Metadaten der Regel vermerkt. Die dort in „req“ aufgeführten Platzhalter müssen, in einem Wort bereits erzeugt worden sein, damit Regel (1) auf P_1 angewendet werden kann.

Lebensmarker

In Kapitel 3.4 wurde beschrieben, dass in allen Regeln einer Workflow Grammatik entweder ein Lebensmarker oder ein Killmarker (im Folgenden auch als Terminierungsmarker bezeichnet) auf beiden Seiten der Regel vorkommt. Dem Prozess sind genau ein Lebensmarker und ein Terminierungsmarker (Killmarker) zugeordnet. Der Lebensmarker identifiziert genau die Regeln, welche während der normalen Ausführung eines Prozesses anzuwenden sind. Der Terminierungsmarker beschreibt dagegen die Regeln, die während der Terminierung eines Prozesses anzuwenden sind.

Die Prozessstrukturregeln einer Workflow Grammatik enthalten, bis auf wenige Ausnahmen, ausschließlich Lebensmarker. Der Terminierungsmarker wird dagegen, wiederum bis auf wenige Ausnahmen, nur in generischen Regeln angewendet. Aus diesem Grund kann der Lebensmarker aus allen Regeln entfernt werden. Die genaue Umsetzung der Terminierung eines Prozesses wird in Kapitel 6.1.3 beschrieben.

6.1.2. Zuordnung von zu Regeln und Aktionen

Regeln von Workflow Grammatiken können, wie bereits erwähnt, bestimmte Aktionen wie beispielsweise den Aufruf eines Web Services oder die Manipulation von Prozessdaten repräsentieren. Die Regeln einer Workflow Grammatik müssen, um festzustellen wann welche Aktion

ausgeführt werden muss, von der Workflow Engine interpretiert werden. In der internen Repräsentation einer Workflow Grammatik können jeder Produktionsregel explizite Aktionen zugeordnet sein. Diese Aktionen müssen genau dann ausgeführt werden, wenn die entsprechende Regel auf ein Teilwort angewendet wird.

Einige der mit den Regeln assoziierten Aktionen werden durch mehrere Regeln beschrieben. So gehört zu einer Aktivierungsregel einer bestimmten Aktivität immer auch mindestens eine Beendigungsregel. Derartige Zusammenhänge zwischen einzelnen Regeln werden für eine effiziente Ausführung in der internen Repräsentation einer Workflow Grammatik explizit modelliert. So werden beispielsweise alle möglichen Beendigungsregeln einer Aktivität in einer Regelgruppe zusammengefasst.

Im Folgenden werden unterschiedliche Regeltypen vorgestellt. Es wird beschrieben, wie sie im Allgemeinen aufgebaut sind, welche konkreten Aktionen sie repräsentieren, und wie sie in der internen Repräsentation der Grammatik dargestellt und organisiert werden.

Aktivierung und Beendigung von Aktivitäten

Aktivitäten werden in Workflow Grammatiken durch Nichtterminale vom Typ Call (C), In (I), Out (O), Assign (A), Wait (W), Evaluation (E) oder Quit (Q) repräsentiert. Diese Nichtterminal Typen wurden bereits in Kapitel 3.3 eingeführt und näher beschrieben. Zu jedem dieser Aktivitätstypen gibt es sowohl Aktivierungs- als auch Beendigungsregeln. Bei einer Aktivierungsregel kommt das betreffende Nichtterminal nur auf der rechten Seite vor, es wird in dieser Regel also erzeugt. Bei einer Beendigungsregel kommt das betreffende Nichtterminale nur auf der linken Seite vor, auf der rechten Seite muss dann stattdessen immer das dazugehörige Terminal vorkommen. Sowohl in der Aktivierungs- als auch in der Beendigungsregel können Variablen vorkommen. Die in der Aktivierungsregel vorkommenden Variablen beschreiben die Eingabeparameter für die entsprechende Aktivität. Die in der Beendigungsregel vorkommenden Variablen beschreiben die Ausgabeparameter der Aktivität.

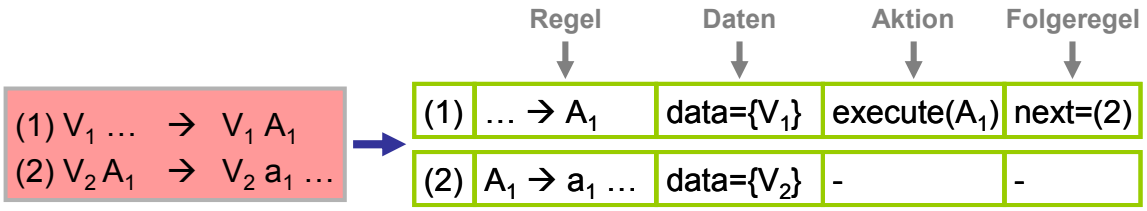
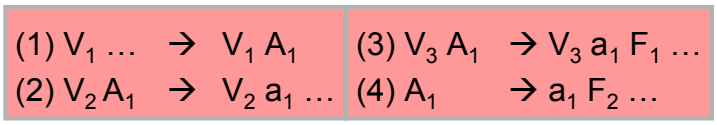


Abbildung 40: Aktivierungs- und Beendigungsregel

Aktivierungs- und Beendigungsregel werden in der internen Repräsentation von Workflow Grammatiken direkt miteinander verknüpft. In Abbildung 40 ist die Transformation einer Aktivierungs- sowie einer Beendigungsregel in die interne Repräsentation an einem Beispiel dargestellt. Die in Regel (1) enthaltenen Variablen werden, wie bereits beschrieben, aus der Regel entfernt und separat dargestellt. Die Information, dass die Regel die Aktivierung der Aktivität A_1 beschreibt, wird explizit dargestellt. Zusätzlich wird die Regel mit der Information angereichert, dass nach der Beendigung der Aktivität A_1 die Regel (2) anzuwenden ist.

Zu jedem Aktivitäts- Nichtterminal kann es mehr als eine Beendigungsregel geben. Aktivitäten können während ihrer Ausführung Fehler erzeugen. Die im Beispiel dargestellte Beendigungsregel beschreibt den Fall, dass die Aktivität erfolgreich, also ohne Fehler beendet wird. In Workflow Grammatiken werden Fehler durch Fehler Nichtterminale repräsentiert. Zu jeder Aktivität gibt es genau eine Beendigungsregel ohne Fehler Nichtterminal und beliebig viele weitere Beendigungsregel mit jeweils einem Fehler Nichtterminal. Es darf dabei pro Aktivität für einen Fehlertypen maximal eine

Beendigungsregel geben. In der internen Repräsentation von Workflow Grammatiken werden alle Beendigungsregeln jeder Aktivität gemeinsam in einer Regelgruppe verwaltet.



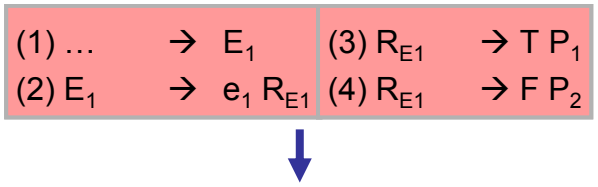
(1)	… → A_1	data={V_1}	execute(A_1)	next[]=(2), next[F_1]=(3) , next[F_2]=(4)
(2)	A_1 → a_1 …	data={V_2}	-	-
(3)	A_1 → a_1 …	data={V_3}	-	-
(4)	A_1 → a_1 …	data={}	-	-

Abbildung 41: Beendigungsregeln mit Fehler- Nichtterminalen

In Abbildung 41 ist ein Ausschnitt aus einer Beispielgrammatik dargestellt. Für die Aktivität A_1 sind mehrere Beendigungsregeln, abhängig vom Auftreten von Fehlern, angegeben. Die Information, welche Regel in welchem Fall anzuwenden ist, wird bereits in der Aktivierungsregel (1) gesammelt und strukturiert hinterlegt. Die Fehler- Nichtterminale werden also aus den einzelnen Regeln extrahiert. Die für die Ausführung relevanten Informationen werden explizit dargestellt.

Kontrollfluss mit booleschen Bedingungen

Bedingter Kontrollfluss basierend auf booleschen Bedingungen wird in Workflow Grammatiken immer durch ein Nichtterminal vom Typ Evaluation modelliert. Dieses Aktivitäts- Nichtterminal wird zunächst, wie jede andere Aktivität auch, erzeugt und repräsentiert damit die Aktivierung der entsprechenden Evaluation Aktivität. In der dazugehörigen Beendigungsregel wird für eine Evaluation Aktivität immer genau ein Nichtterminal vom Typ Ergebnisvariable (R) erzeugt. Es repräsentiert in diesem Fall eine boolesche Variable, die das Ergebnis der Evaluation Aktivität enthält und deren Wert nach außen sichtbar ist. Zu diesem Nichtterminal gibt es genau zwei weitere Regeln. Auf der linken Seit enthalten sie das Nichtterminal vom Typ Ergebnisvariable, auf der rechten Seite kommt entweder ein Nichtterminal vom Typ T („true“) oder eines vom Typ F („false“) vor.



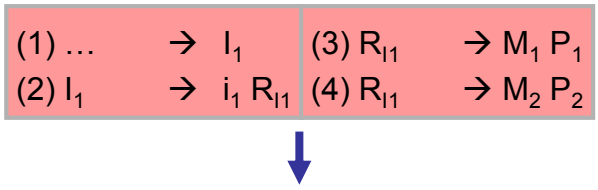
(1)	… → E_1	data={V_1}	execute(E_1)	next=(2)
(2)	E_1 → e_1 R_{E1}	data={R_{E1}}	-	next[T]=(3), next[F]=(4)
(3)	R_{E1} → P_1	data={R_{E1}}	-	-
(4)	R_{E1} → P_2	data={R_{E1}}	-	-

Abbildung 42: Auswertung boolescher Bedingungen

Die Nichtterminale vom Typ T und F signalisieren, welche Regel, abhängig vom konkreten Wert der Ergebnisvariablen, angewendet werden soll. Die Beendigungsregel einer Evaluation Aktivität sowie die soeben beschriebenen darauf folgenden Auswertungsregeln werden in der internen Repräsentation von Workflow Grammatiken miteinander verknüpft und gemeinsam verwaltet. Abbildung 42 zeigt ein Beispiel für eine solche interne Darstellung. In der Beendigungsregel (2) der Evaluation Aktivität E_1 ist bereits vermerkt, welche Regel, abhängig vom in der Ergebnisvariablen R_{E1} hinterlegten Wert, anschließend angewendet werden soll.

Kontrollfluss abhängig von empfangenen Nachrichten

Von Nachrichten abhängiger Kontrollfluss kann nach der Ausführung einer In Aktivität modelliert werden. In Workflow Grammatiken repräsentiert eine In Aktivität den Empfang genau einer Nachricht über eine von mehreren möglichen Schnittstellen. Abhängig davon, über welche der durch die In Aktivität angebotenen Schnittstellen eine Nachricht empfangen wurde, können unterschiedliche Folgeregeln modelliert werden. Ähnlich wie im Falle des bedingten Kontrollflusses abhängig von Fehlern oder von booleschen Bedingungen werden die Regeln auch hier bereits vor der Ausführung interpretiert, verändert, und durch die dabei gewonnenen Informationen angereichert.



(1)	… → I_1	data={V_1}	execute(I_1)	next=(2)
(2)	I_1 → i_1 R_{I1}	data={R_{I1}}	-	next[M_1]=(3), next[M_2]=(4)
(3)	R_{I1} → P_1	data={R_{I1}}	-	-
(4)	R_{I1} → P_2	data={R_{I1}}	-	-

Abbildung 43: Empfang einer Nachricht, bedingter Kontrollfluss

In Abbildung 43 ist die Beendigungsregel sowie die darauf folgenden Auswertungsregeln der In Aktivität I_1 dargestellt. Die Nichtterminale M_1 und M_2 repräsentieren die beiden Schnittstellen, über welche die Aktivität I_1 eine Nachricht empfangen kann. In der Ergebnisvariable R_{I1} werden sowohl die empfangene Nachricht als auch die Information über die Schnittstelle, über welche sie empfangen wurde, zwischengespeichert.

Verwaltung von Scopes

Scopes stellen in Workflow Grammatiken Lebensbereichen von Variablen dar. Bei der Erzeugung eines Scopes werden seine linke und rechte Grenze, alle darin lebenden Variablen, sowie optional Platzhalter erzeugt. Zu jedem Scope gibt es eine Beendigungsregel, welche den Scope und alle darin lebenden Variablen löscht. Die generischen Regeln von Workflow Grammatiken beschreiben, dass die Beendigungsregel eines Scopes erst dann angewendet werden darf, wenn er nur noch Variablen und Terminalzeichen enthält.

Die Aktivierungs- sowie Beendigungsregel eines Scopes werden in der internen Repräsentation in einer Regel zusammengefasst. Die Grenzen des Scopes sowie die darin enthaltenen Variablen werden aus der Regel entfernt. Im Gegenzug dazu wird in den Metadaten der Regel vermerkt, dass die Anwendung dieser Regel einen entsprechenden Scope mit dazugehörigen Variablen erzeugt. Die Grenzen des Scopes sowie die darin lebenden Variablen werden zur Laufzeit von der Workflow

Engine verwaltet. Sie ist ebenfalls für die abschließende Beendigung des Scopes verantwortlich (siehe auch Kapitel 6.1.3).

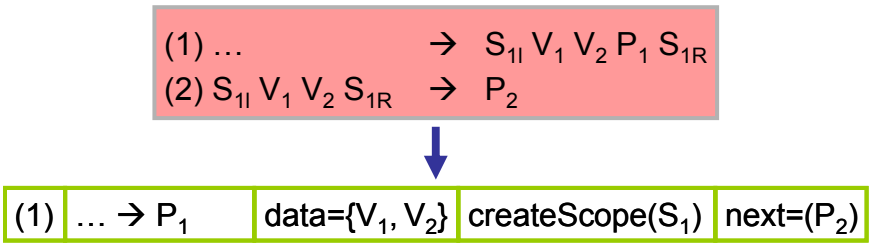


Abbildung 44: Erzeugen von Scopes

In Abbildung 44 ist die Abbildung von Aktivierungs- sowie Beendigungsregel eines Scopes auf die interne Repräsentation beispielhaft dargestellt. Wie bereits erwähnt, werden die Variablen sowie die Grenzen des Scopes aus der Regel entfernt. Die Aktion „createScope(S_1)“ bedeutet, dass mit Anwendung dieser Regel der Scope S_1 erzeugt wird. Die unter „data“ angegebenen Variablen V_1 und V_2 sowie der auf der rechten Seite der Regel erzeugte Platzhalter P_1 leben innerhalb dieses Scopes. Anstatt des Verweises auf eine Folgeregel wird unter „next“ direkt angegeben, dass beim Beenden des Scopes der Platzhalter P_2 erzeugt werden sollen.

Mobile Platzhalter

Regeln von Workflow Grammatiken können sogenannte mobile Platzhalter erzeugen. Diese Platzhalter sind Nichtterminale, welche sich frei im gesamten Wort bewegen können. Werden mobile Platzhalter in einer Regel auf der linken Seite benötigt, wie beispielsweise bei der bereits beschriebenen Modellierung eines Joins, dann ist ihre aktuelle Position im Wort nicht relevant. Da sie frei beweglich sind, können sie jederzeit an jede Stelle im Wort bewegt werden.

In der internen Repräsentation eines Workflow Grammatik werden mobile Platzhalter aus alle Regen entfernt. Sie werden interpretiert als globale Zählvariablen. In den Regeln, in denen ein mobiler Platzhalter auf der rechten Seite erzeugt wird, muss der entsprechende globale Zähler um eins erhöht werden. In den Regeln, in denen mobile Platzhalter auf der linken Seite benötigt werden, muss der Wert des Zählers überprüft werden.

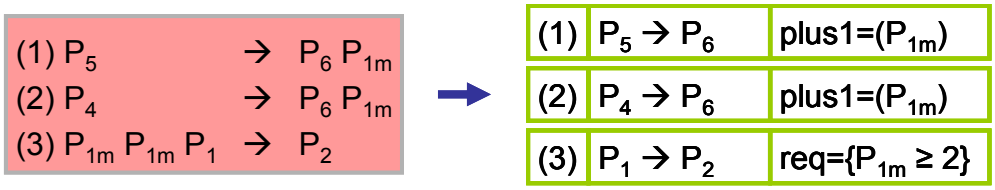


Abbildung 45: Erzeugen von mobilen Platzhaltern

In Abbildung 45 sind Regeln einer Grammatik dargestellt, links in der Ausgangsform, rechts die interne Repräsentation. Regel (1) und Regel (2) erzeugen jeweils einen mobilen Platzhalter P_{1m}. Der Platzhalter wird aus diesen Regeln entfernt. Zusätzlich wird in den Metadaten dieser Regeln vermerkt, dass der globale Zähler für den mobilen Platzhalter P_{1m} um eins erhöht werden muss. Regel (3) benötigt zur Ausführung zwei mobile Platzhalter P_{1m}. Die Platzhalter werden aus dieser Regel ebenfalls entfernt. In den Metadaten der Regel wird vermerkt, dass eine Anwendung dieser Regel als Vorbedingung verlangt, dass der globale Zähler für mobile Platzhalter P_{1m} mindestens den Wert zwei besitzt. Wird diese Regel angewendet, dann wird dieser Zähler um zwei erniedrigt.

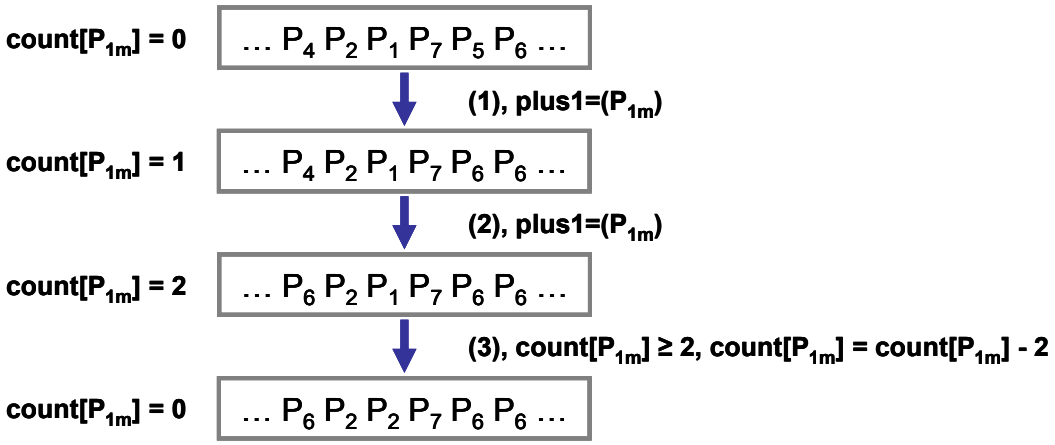


Abbildung 46: Mobile Platzhalter als globale Zähler

Ein Beispiel für die Umsetzung von mobilen Platzhaltern als globale Zähler ist in Abbildung 46 gezeigt. Die Abbildung beschreibt, von oben nach unten, einen Teil einer Ableitung anhand der in zuvor Abbildung 45 gezeigten Grammatikregeln. Zu jedem Schritt ist links der aktuelle Stand des globalen Zählers für den mobilen Platzhalter P_{1m} angegeben. Auf der rechten Seite ist angegeben, welche Regeln auf das jeweilige Teilwort angewendet wurden und welche Aktionen damit verbunden sind. In den ersten beiden Schritten wird der globale Zähler um jeweils eins erhöht, dies entspricht dem Erzeugen von mobilen Platzhaltern P_{1m}. Im letzten Schritt wird der Zähler dagegen um zwei erniedrigt. Dies repräsentiert, dass die angewendete Regel (3) zwei mobile Platzhalter löscht.

6.1.3. Behandlung der generischen Regeln

Die Regelmenge einer Workflow Grammatik wird unterschieden in Prozessstrukturregeln und generische Regeln. Die Prozessstrukturregeln beschreiben die eigentliche Struktur des durch die Grammatik beschriebenen Prozesses. Die generischen Regeln modellieren dagegen allgemeines Verhalten, welches für jedes Prozessmodell gültig ist. Beim Erzeugen einer konkreten Workflow Grammatik werden die generischen Regeln im Allgemeinen nicht explizit formuliert. Das bedeutet für die Ausführung solcher Grammatiken, dass die generischen Regeln direkt implementiert werden müssen. Das Wissen über diese Regeln wird vorausgesetzt. Im Folgenden werden die unterschiedlichen Arten von generischen Regeln und ihre Behandlung durch eine Workflow Grammatik Workflow Engine beschrieben.

Lebensdauer und Sichtbarkeit von Variablen

Lebensdauer und Sichtbarkeit einer Variablen sind in Workflow Grammatiken immer an einen Scope gebunden. Die entsprechenden generischen beschreiben, dass jede Variable innerhalb ihres Scopes frei beweglich ist. Es wurde bereits beschrieben, dass sowohl Variablen als auch Scopes in der internen Repräsentation einer Grammatik aus den Regeln entfernt und in den Metadaten gespeichert werden. Sowohl Variablen als auch Scopes werden zur Laufzeit nicht direkt im Teilwort sondern getrennt davon verwaltet. Dabei wird auch das durch die generischen Regeln beschriebene Verhalten umgesetzt.

Lebensdauer von Scopes

Die generischen Regeln für Scopes modellieren, dass ein Scope genau dann beendet werden kann, wenn er nur noch die in ihm lebenden Variablen und Terminale enthält. Es wurde bereits beschrieben, dass Scopes nicht direkt im Wort verwaltet werden. Zur Ausführung ist die Workflow Engine dafür verantwortlich, den Inhalt eines Scopes zu überwachen. Besteht der entsprechende Bereich des

Teilwortes nur noch aus Terminalen (Variablen werden außerhalb des Wortes verwaltet), so kann der Scope und alle in ihm lebenden Variablen gelöscht werden.

Vorzeitiges Prozessende

Das Erzeugen eines Nichtterminals vom Typ Killmarker signalisiert das vorzeitige und sofortige Beenden einer Prozessinstanz. Dies wird durch entsprechende generische Regeln modelliert. Die Umsetzung dieser Regeln bedeutet, dass das Anwenden einer Regel, in welcher ein Killmarker erzeugt wird, das Ende der Prozessinstanz bedeutet. Alle noch laufenden Aktivitäten werden sofort beendet.

6.1.4. Zusammenfassung

In den vorherigen Kapiteln wurde beschrieben, wie eine gegebene Workflow Grammatik in eine für die Ausführung geeignete interne Repräsentation überführt werden kann. Der Grundaufbau dieser internen Repräsentation entspricht dem einer kontextfreien Grammatik. Die einzelnen Regeln der Grammatik werden bei der Transformation mit Metadaten angereichert. Diese Metadaten werden für eine anschließende Ausführung benötigt.

In Kapitel 6.1.1 wurde zunächst gezeigt, wie eine gegebene Workflow Grammatik in eine kontextfreie Grammatik umgewandelt werden kann. Durch die Entfernung von Nichtterminalen aus der linken Seite einer Regel gehen im Allgemeinen Informationen verloren. Diese werden in den Metadaten der entsprechenden Regel hinterlegt. Die Hauptquellen für Kontextfreiheit in Workflow Grammatiken sind die Verwendung von Variablen zur Modellierung von Datenfluss sowie die Verwendung mobiler Platzhalter zur Realisierung von Synchronisation.

In Kapitel 6.1.2 wurde anschließend die Semantik einzelner Regeln einer Workflow Grammatik betrachtet. Regeln können Aktionen wie beispielsweise die Aktivierung von Aktivitäten repräsentieren. Die einzelnen Regeln wurden interpretiert, die durch sie dargestellten Aktionen wurden in den Metadaten dieser Regeln explizit hinterlegt. Regeln, welche im Bezug auf die Ausführung gemeinsam verwendet werden, wurden zu Gruppen zusammengefasst und miteinander verknüpft. Die so geschaffenen Strukturen erleichtern die Ausführung von Workflow Grammatik in ihrer internen Repräsentation.

In Kapitel 6.1.3 wurde schließlich beschrieben, wie die generischen Regeln von Workflow Grammatiken behandelt werden. Eine Workflow Grammatik, wie sie der Workflow Engine übergeben wird, ist nicht vollständig. Jede solche Grammatik enthält implizit noch weitere Regeln, die sogenannten generischen Regeln. Diese beschreiben allgemeines Verhalten, wie beispielsweise die Sichtbarkeit und Lebensdauer von Variabeln, beeinflussen aber nicht die eigentlich in einer Workflow Grammatik modellierte Prozessstruktur.

6.2. Aufbau einer Workflow Engine für Workflow Grammatiken

Der allgemeine Aufbau einer Workflow Engine und ihre Unterteilung in einzelne Komponenten wurden bereits in Kapitel 5.3 vorgestellt. Der Entwurf einer Workflow Engine für Workflow Grammatiken orientiert sich an diesem modularen Aufbau. In diesem Kapitel wird für die einzelnen Komponenten einer generischen Workflow Engine jeweils beschrieben, wie ihre konkrete Ausgestaltung für die Ausführung von Workflow Grammatiken ist. Die Kernkomponente ist dabei der in Kapitel 6.2.1 beschriebenen Navigator, er setzt die in Kapitel 4 allgemein beschriebenen und diskutierten Verfahren zum Erzeugen von Wörtern aus einer Grammatik um.

6.2.1. Navigator

Workflow Grammatiken werden, wie in Kapitel 6.1 beschrieben, vor der Ausführung in kontextfreie Grammatiken umgewandelt. In Kapitel 4.2 wurde ein Verfahren vorgestellt, mit dem unter Beachtung der speziellen Anforderung von Workflow Grammatiken aus einer kontextfreien Grammatik Wörter produziert werden können. Der Navigator einer Workflow Grammatik Workflow Engine implementiert dieses Verfahren.

6.2.2. Prozessmodellspeicher

Workflow Grammatiken beschreiben Prozesse durch Grammatiken. Der Prozessmodellspeicher ist in diesem Fall also ein Speicher für Grammatiken. Im vorherigen Kapitel 6.1 wurde beschrieben, dass Workflow Grammatiken vor ihrer Ausführung in eine interne Repräsentation überführt werden. Diese wird vom Navigator für die Ausführung verwendet. Im Prozessmodellspeicher sind also sowohl die ursprüngliche Workflow Grammatik als auch ihre transformierter interne Repräsentation hinterlegt. Die interne Repräsentation wird dem Navigator zur Verfügung gestellt.

Der Prozessmodellspeicher verwaltet die ursprüngliche sowie die interne Repräsentation einer Workflow Grammatik gemeinsam. Die Umwandlung einer Grammatik verändert nur ihre Regeln. Das Startsymbol, die Menge der Nichtterminale sowie die Menge der Terminale bleiben unverändert und gelten für beide Repräsentationsformen einer Workflow Grammatik.

6.2.3. Instanzspeicher

Bei Workflow Grammatik entspricht die Ausführung eines Prozessmodells dem Erzeugen eines Wortes aus einer Grammatik, also einer Ableitung. Die Ableitung beginnt beim Startsymbol, in jedem Schritt kann das Teilwort durch die Anwendung einer Rege verändert werden. Der aktuelle Zustand einer Prozessinstanz wird durch das jeweils aktuelle Teilwort repräsentiert.

Der Instanzspeicher ist in diesem Fall ein Speicher für Teilworte von Ableitungen. Ein Teilwort ist eine geordnete Folge von Terminal- und Nichtterminalinstanzen. Der Ausführungszustand einzelner Aktivitäten ist in den entsprechenden Nichtterminalinstanzen hinterlegt. Terminalinstanzen repräsentieren bereits beendete Aktivitäten, der Beendigungszustand ist dort ebenfalls hinterlegt.

6.2.4. Prozessdatenspeicher und Auswertung von Ausdrücken

Die in Workflow Grammatiken verwendeten Datentypen werden durch XML Schema definiert. Variablen sind XML Strukturen, Ausdrücke werden als XPath Ausdrücke angegeben. Der Prozessdatenspeicher muss die Werte von Variablen als XML Strukturen zur Verfügung stellen.

6.2.5. Programmaufruf

Für Kommunikation und den Aufruf von Programmen werden in Workflow Grammatiken Web Services verwendet. Zur Unterstützung komplexer Interaktionsszenarien werden von BPEL spezifizierte Mechanismen wie PartnerLinks oder CorrelationSets verwendet. Die Komponente für den Programmaufruf muss also zum einen Web Service Kommunikation unterstützen und zum anderen auch die genannten zusätzlichen Konstrukte implementieren.

6.2.6. Schnittstellen nach außen

Instanzen von Workflow Grammatiken werden durch den Aufruf von entsprechenden Web Service Schnittstellen erzeugt und gestartet. Auch die Kommunikation mit laufenden Prozessinstanzen geschieht durch Web Service Aufrufe. Die Funktionalität, Web Service Schnittstellen zu instanziieren, wird bereits von der Komponente für den Programmaufruf abgedeckt. Sie kann zusätzlich dafür verwendet werden, die für die Erzeugung von Prozessinstanzen nötigen Schnittstellen anzubieten.

7 Prototyp einer Workflow Engine für Workflow Grammatiken

In diesem Kapitel wird die prototypische Implementierung einer Workflow Engine für die Ausführung von Workflow Grammatiken beschrieben. Der Aufbau der Implementierung orientiert sich an der in Kapitel 5.3 vorgestellten Unterteilung der Funktionalitäten einer Workflow Engine in einzelne Komponenten. Die Ausprägung dieser Komponenten für die Ausführung von Workflow Grammatiken wurde in Kapitel 6.2 beschrieben.

7.1. Dateiformat für Workflow Grammatiken

In [Vuk11] wurde ein XML basiertes Dateiformat für die Darstellung von Workflow Grammatiken entwickelt. Die in dieser Arbeit prototypisch implementierte Workflow Engine bekommt als Eingabe eine Datei mit einer in diesem Dateiformat beschriebene Workflow Grammatik. In Listing 6 ist ein Beispiel für eine einfache Workflow Grammatik dargestellt. Die Elemente „terminalReference“ und „nonTerminalReference“ verweisen mit ihrem Attribut „target“ auf den Wert des Attributes „id“ von Terminalen und Nichtterminale. Diese Referenzierung muss beim Einlesen einer Workflow Grammatik Datei zunächst explizit aufgelöst werden. Zusätzlich zu der Workflow Grammatik XML Datei werden alle in dieser Datei referenzierten WSDL oder XML Schema Dateien benötigt.

```
<wogGrammar xmlns="http://iaas.uni-stuttgart.de/wog/model"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <nonTerminalSet>
    <nonTerminal id="S"
                 xsi:type="tStartSymbol">
      <partnerLinks> … </partnerLinks>
    </nonTerminal>
    <nonTerminal id="A1"
                 xsi:type="tCall"
                 partnerLink="…"
                 operation="…">
    </nonTerminal>
  </nonTerminalSet>
  <terminalSet>
    <terminal id="a1">
      <nonTerminalReference target="A1" />
    </terminal>
  </terminalSet>
  <ruleSet>
    <rule>
      <ruleLeftSide>
        <nonTerminalReference target="S" />
      </ruleLeftSide>
      <ruleRightSide>
        <nonTerminalReference target="A1" />
      </ruleRightSide>
    </rule>
    <rule>
      <ruleLeftSide>
        <nonTerminalReference target="A1" />
```

```
      </ruleLeftSide>
      <ruleRightSide>
        <terminalReference target="a1" />
      </ruleRightSide>
    </rule>
  </ruleSet>
  <startSymbol>
    <nonTerminalReference target="S" />
  </startSymbol>
</wogGrammar>
```

Listing 6: Workflow Grammatik XML Darstellung

7.2. Interne Repräsentation für ausführbare Workflow Grammatiken

In Kapitel 7.1 wurde beschrieben, dass Workflow Grammatiken als XML Dateien vorliegen. Sie werden zunächst eingelesen und anschließend in eine interne Repräsentation überführt. Der prinzipielle Aufbau der internen Repräsentation sowie die Überführung einer gegebenen Workflow Grammatik in diese Form wurde bereits in Kapitel 6.1 ausführlich beschrieben. Der in dieser Arbeit entwickelte Prototyp einer Workflow Engine für Workflow Grammatiken implementiert die dort beschriebenen Umwandlungsschritte.

7.3. Funktionsweise des Navigators

In Kapitel 4.2 wurde ein Verfahren zur Erzeugung von Wörtern aus einer kontextfreien Workflow Grammatik beschrieben. Der Navigator des entwickelten Prototyps einer Workflow Engine für Workflow Grammatiken implementiert das dort beschriebene Verhalten. Der Zustand der Prozessausführung wird durch das aktuelle Teilwort repräsentiert. In jedem Schritt der Ausführung sind zwei Entscheidungen zu treffen. Zunächst muss ausgewählt werden, auf welches der im Teilwort enthaltenen Nichtterminale eine Regel angewendet wird. Anschließend muss entschieden, welche Regel auf das gewählte Nichtterminal angewendet wird.

Die Umsetzung dieses Vorgehens ist in Abbildung 47 in Form eines Sequenzdiagramms dargestellt. Ebenfalls dargestellt sind ein Ausschnitt aus der diesem Beispiel zugrunde liegenden Grammatik und das aktuelle Teilwort. Das aktuelle Teilwort wird im Sequenzdiagramm durch die Komponente „Word“ repräsentiert.

Im ersten Schritt ruft der Navigator die Methode „selectOneNonTerminal()“ der Komponente „NonTerminalSelector“ auf. Diese Komponente implementiert den ersten Auswahlschritt, sie wählt aus dem aktuellen Teilwort genau ein Nichtterminal aus. Im Prototyp erfolgt diese Auswahl nach einer einfachen Logik, es wird das erste gefundene Nichtterminal ausgewählt. Es ist aber denkbar, in dieser Komponente erweiterte Logik zu implementieren. Beispielsweise können Nichtterminale, welche die Ausführung lang laufender Aktivitäten auslösen, bevorzugt ausgewählt werden. Damit kann die Ausführung eines Prozesses beschleunigt werden. Die Komponente „NonTerminalSelector“ gibt als Ergebnis genau ein Nichtterminal zurück.

Im nächsten Schritt wird die Methode „getRules(…)“ der Grammatik aufgerufen und dabei das soeben gewählte Nichtterminal übergeben. Es werden alle Regeln bestimmt, die auf das übergebene Nichtterminal angewendet werden können. Das Ergebnis ist eine Menge von Produktionsregeln, die auf ihrer linken Seite genau das übergebene Nichtterminal enthalten.

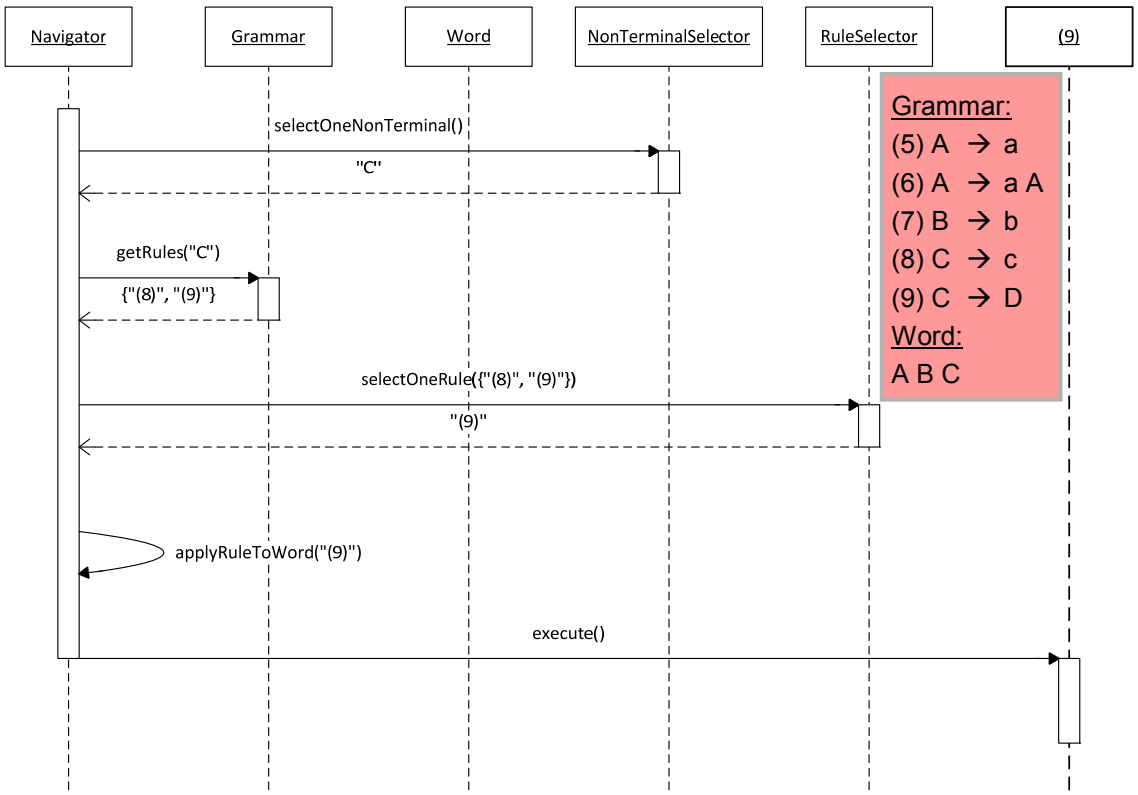


Abbildung 47: Sequenzdiagramm Regelauswahl

Die Menge aller möglichen anzuwendenden Regeln wird der Methode „selectOneRule(…)“ der Komponente „RuleSelector“ übergeben. Sie implementiert den zweiten Auswahlschritt und wählt genau eine der übergebenen Regeln aus. Die Auswahl zwischen mehreren Regeln erfolgt anhand des aktuellen Prozesszustandes. In Workflow Grammatiken kann die Wahl zwischen alternativen Regeln für das gleiche Nichtterminal immer deterministisch getroffen werden. Sie ist abhängig von Werten, die in Ergebnisvariablen hinterlegt sind, oder vom Typ aufgetretener Fehler (siehe dazu Kapitel 3.4). Als Ergebnis wird genau eine Regel zurückgegeben.

Nachdem bestimmt wurde, welche Regel im nächsten Schritt angewendet werden soll, wird diese zunächst auf das aktuelle Teilwort angewendet. In diesem Schritt werden durch die Methode „applyRuleToWord(…)“ lediglich Elemente des Teilwortes manipuliert. Werden durch die Anwendung der Regel auf das Teilwort neue Aktivitäts- Nichtterminale erzeugt, dann werden diese zunächst als blockiert markiert. Aktivitäts- Nichtterminale dürfen erst dann weiter verarbeitet (also durch Regeln ersetzt) werden, wenn die mit ihnen assoziierte Aktivität beendet wurde.

Im letzten Schritt wird die Methode „execute()“ der gewählten Regel aufgerufen. Sie führt die mit der Regel assoziierten Aktionen durch.

Bei der Ausführung der mit einer Produktionsregel assoziierten Aktionen werden zwei Fälle unterschieden. Aktionen, welche im Allgemeinen schnell durchgeführt werden können, werden vom Navigator synchron aufgerufen. Er kann die Prozessausführung in dieser Zeit nicht fortsetzen. Die Funktionsweise des synchronen Aufrufs ist in Abbildung 48 als Sequenzdiagramm dargestellt. Die Ausführung der mit einer Regel assoziierten Aktionen erfolgt immer über den Aufruf der Methode „execute()“ der entsprechenden Regel. Im synchronen Fall führt die Regel die betreffenden Aktionen, im Beispiel eine Zuweisung, direkt aus. Nach der Ausführung der Aktivität wird das Nichtterminal, welches diese im Teilwort repräsentiert, als nicht mehr blockiert markiert. Damit ist es möglich, Regeln auf dieses Nichtterminal anzuwenden. Sind alle mit einer Regel assoziierten Aktionen ausgeführt, geht die Kontrolle an den Navigator zurück.

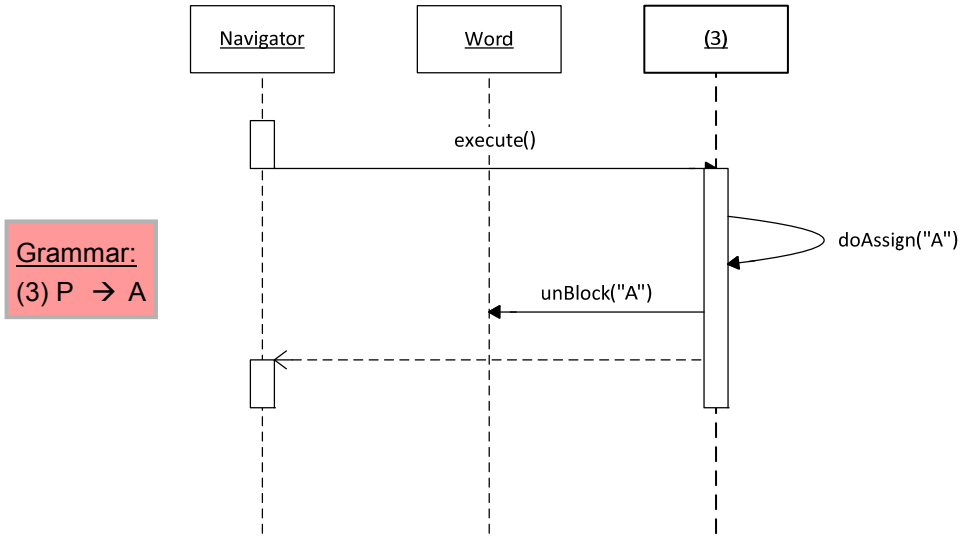


Abbildung 48: Sequenzdiagramm synchrone Aktivität

Sind die mit einer Regel assoziierten Aktionen lang laufend, dann soll vermieden werden, dass der Navigator für die Zeit der Ausführung dieser Aktivitäten blockiert ist. Der im hier beschriebenen Prototypen implementierte Mechanismus zur asynchronen Ausführung von Aktivitäten ist in Abbildung 49 als Sequenzdiagramm dargestellt.

Im ersten Schritt wird, wie auch im zuvor beschriebenen synchronen Fall, die Methode „execute(…)“ der auszuführenden Regel aufgerufen. Diese Methode führt die mit der Regel assoziierte Aktivität nicht selbst aus. Sie erzeugt ein Objekt, welches die Ausführung der Aktivität übernimmt. Im gezeigten Beispiel ist dies der Aufruf eines Web Services, der durch die Komponente „WebServiceCall_C“ implementiert wird. Nach der Erzeugung dieser Komponente wird sie der Komponente „ExecutorService“ zur Ausführung übergeben. Die Komponente „ExecutorService“ verwaltet einen Pool von Threads und führt die übergebenen Objekte mit dessen Hilfe aus. Nachdem das Objekt „WebServiceCall_C“ erzeugt und dem „ExecutorService“ übergeben wurde, geht die Kontrolle an den Navigator zurück. Er kann nun, während parallel noch Aktivitäten ausgeführt werden, mit der Anwendung weiterer Regeln auf das Teilwort fortfahren.

Im in Abbildung 49 gezeigten Beispiel ist noch ein weiterer wichtiger Aspekt der Ausführung enthalten. Während der Ausführung kann die Situation eintreten, dass alle im Teilwort enthaltenen Nichtterminale blockiert sind. In diesem Fall muss der Navigator die Ausführung so lange unterbrechen, bis die Blockade eines dieser Nichtterminale aufgehoben wird. Stellt der Navigator also fest, dass er keine Regeln mehr auf das Wort anwenden kann, die Prozessausführung aber auch noch nicht beendet worden ist, dann versetzt er sich selbst in den Ruhezustand. Zuvor ruft er die Methode „notifyMeWhenWorkIsAvailable“ der Komponente „Word“ auf. Damit gibt er bekannt, dass er sich in den Ruhezustand versetzt und aufgeweckt werden muss, wenn die Prozessausführung fortgesetzt werden kann. Mit dem Aufruf der Methode „wait()“ versetzt sich der Navigator anschließend in den Ruhezustand.

Wenn eine asynchron ausgeführte Aktivität beendet wird, dann wird das Nichtterminal, welches diese Aktivität im aktuellen Teilwort repräsentiert, durch den Aufruf der Methode „unBlock(…)“ als nicht mehr blockiert markiert. Befindet sich der Navigator zu diesem Zeitpunkt im Ruhezustand, so wird er anschließend durch de Aufruf der Methode „notify()“ wieder aktiviert.

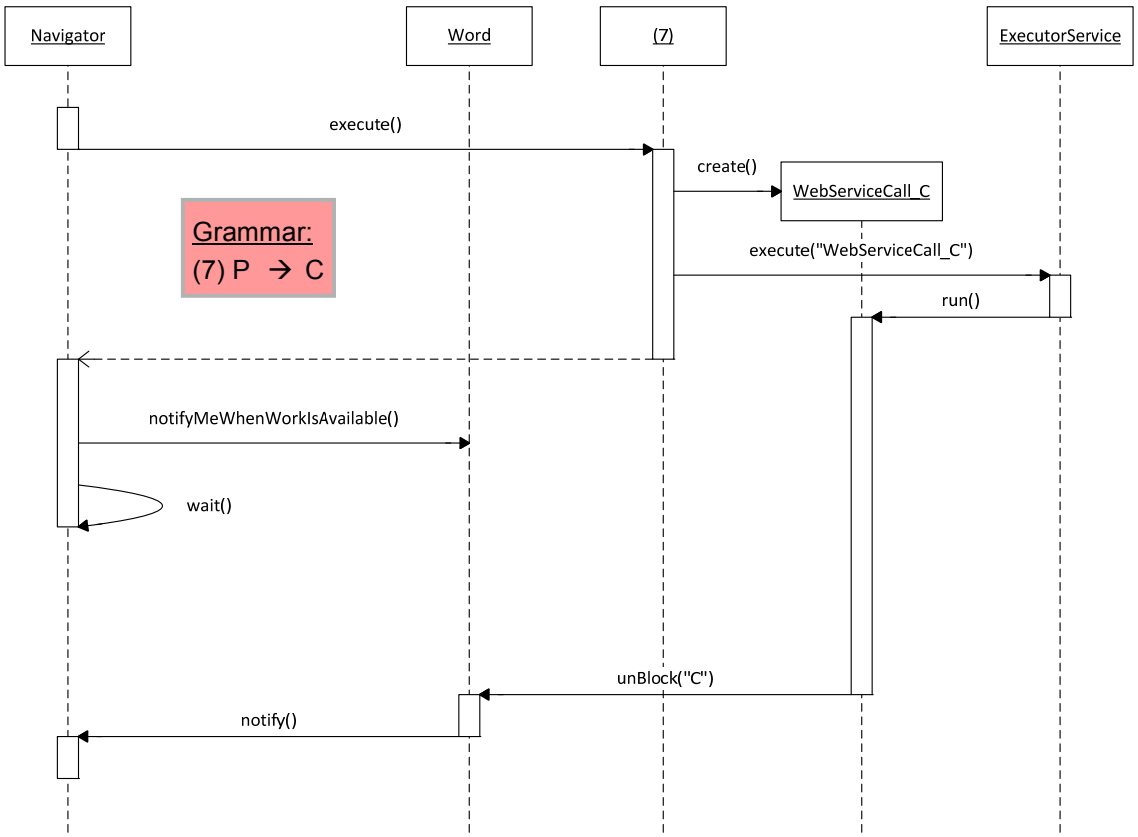


Abbildung 49: Sequenzdiagramm asynchrone Aktivität

7.4. Weitere Komponenten der Workflow Engine

Die Programmaufrufkomponente einer Workflow Engine für Workflow Grammatiken muss Web Service Kommunikation unterstützen. Der entwickelte Prototyp ist in der Lage, einfache Web Service Aufrufe durchzuführen. Die in Workflow Grammatiken verwendeten erweiterten Konzepte von BPEL, wie beispielsweise „CorrelationSets“ oder „MessageExchange“ werden nicht beachtet.

Die Datentypen von Variablen werden in Workflow Grammatiken durch XML Schema definiert. Workflow Grammatiken definieren, wiederum angelehnt an BPEL, verschiedene Möglichkeiten für die Definition von Ausdrücken und Zuweisungen. Der implementierte Prototyp unterstützt lediglich einen Teil der in Workflow Grammatiken möglichen Varianten für Ausdrücke und Zuweisungen.

Der Prozesszustand wird in Workflow Grammatiken durch das aktuelle Teilwort repräsentiert. Der implementierte Prototyp speichert den Zustand der Ausführung und der Variablen nicht persistent ab. Er ist zudem nicht in der Lage, mehrere Prozesse parallel auszuführen

7.5. Zusammenfassung

In diesem Kapitel wurde die Implementierung eines Prototyps einer Workflow Engine für die Ausführung von Workflow Grammatiken beschrieben. Die dafür wesentlichen Prinzipien und Verfahren wurden bereits in vorherigen Kapiteln dieser Arbeit ausführlich behandelt. Der allgemeine Ausbau einer Workflow Engine wurde in Kapitel 5 betrachtet. Das grundlegende Verfahren zum Erzeugen eines Wortes aus einer Grammatik wurde in Kapitel 4 entwickelt. Die Interpretation einer Workflow Grammatik sowie ihre Überführung in eine für die Ausführung aufbereitete interne Repräsentation wurde in Kapitel 6 vorgestellt.

In 7.1 wurde zunächst das XML Dateiformat für Workflow Grammatiken kurz vorgestellt. In Kapitel 7.2 wurde die Überführung einer als XML Datei gegebenen Workflow Grammatik in die interne Repräsentation nochmals kurz angesprochen. In Kapitel 7.3 wurden dann ausgewählte Aspekte der Implementierung des Navigators anhand von Beispiel Sequenzdiagrammen vorgestellt. In Kapitel 7.4 wurde abschließend die Implementierung der weiteren Komponenten der Workflow Engine kurz beschrieben.

8 Zusammenfassung

Im Folgenden wird die gesamte Arbeit noch einmal zusammengefasst. Neben der Beschreibung der behandelten Themen wird ebenfalls festgehalten, welche Aspekte der betrachteten Themen in dieser Arbeit nicht behandelt wurden. Es wird ein Ausblick gegeben, zu welchen Themen weitere Arbeiten möglich sind.

Der inhaltliche Aufbau dieser Arbeit wird in Abbildung 50 dargestellt. Die beiden wichtigsten Grundlagen für diese Arbeit sind die in Kapitel 2 vorgestellte Theorie der Formalen Sprachen, Grammatiken und Automatenmodelle sowie die in Kapitel 3 vorgestellten Workflow Grammatiken. Die Darstellung mit gestrichelten Linien deutet an, dass in diesen Kapiteln nichts wesentlich Neues erarbeitet wird. Es werden bekannte und in anderen Arbeiten bereits bearbeitete Themen vorgestellt und zusammengefasst. Der Fokus liegt dabei immer auf den für diese Arbeit relevanten Aspekten der betrachteten Themen.

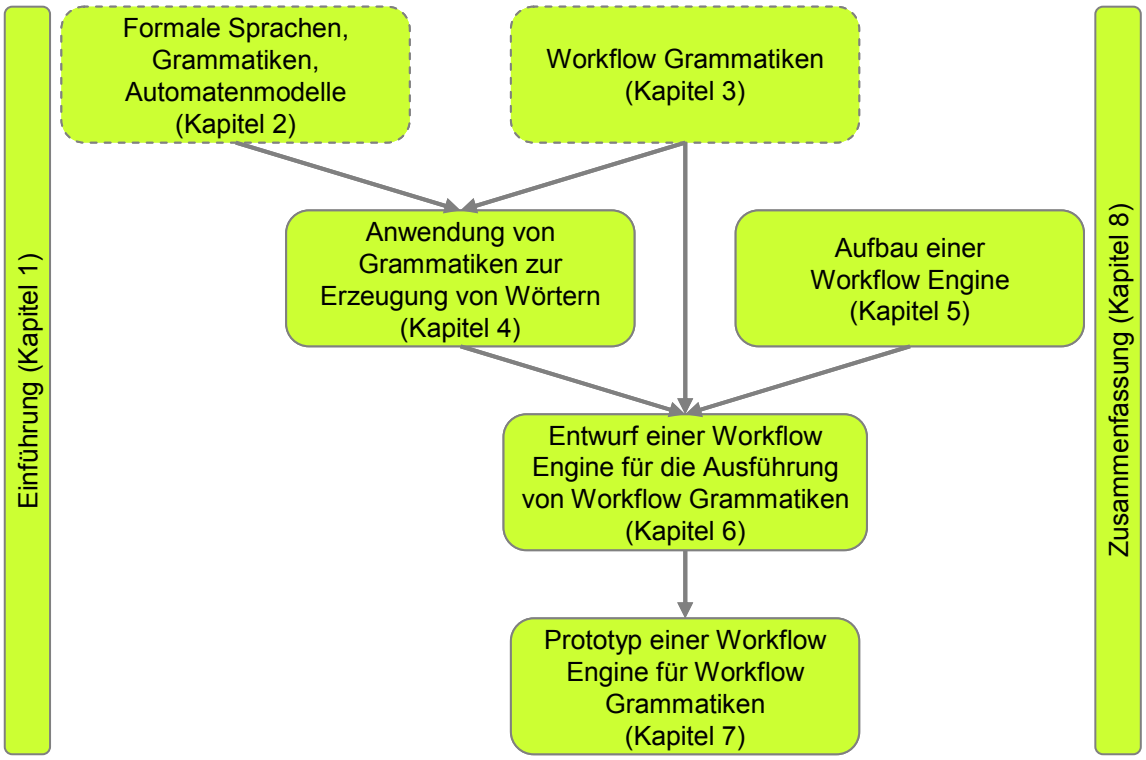


Abbildung 50: Überblick über die gesamte Arbeit

Das grundlegende Thema dieser Arbeit ist die Ausführung von Workflow Grammatiken. Das Vorgehen bei der Ausführung einer Workflow Grammatik entspricht dem Erzeugen eines Wortes aus einer Grammatik. Dieser Zusammenhang motiviert Kapitel 4. In diesem Kapitel werden allgemeine Verfahren zum Erzeugen von Wörtern aus Grammatiken entwickelt. Parallel dazu werden die entworfenen Verfahren auf ihre Anwendbarkeit auf Workflow Grammatiken überprüft und, wenn nötig, daran angepasst. Es werden zwei Forderungen hergeleitet und formuliert, die jedes Verfahren zur Erzeugung von Wörtern aus Workflow Grammatiken erfüllen muss. Der Entwurf der einzelnen Verfahren zum Erzeugen von Wörtern orientiert sich an den in Kapitel 2 vorgestellten Verfahren zum Erkennen von Wörtern.

Die vorgestellten Verfahren zur Erzeugen von Wörtern einer Grammatik orientieren sich an Methoden und Modellen aus der Theoretischen Informatik. Basierend darauf können in weiteren Arbeiten formale Untersuchungen zu den vorgestellten Verfahren vorgenommen werden. Die Korrektheit ihrer Konstruktion, die Komplexität oder ihr Bezug zu den Klassen der Chomsky Hierarchie sind mögliche Themen.

In Kapitel 5 werden zunächst verschiedene Architekturen bestehender Workflow Engines betrachtet. Sie werden vor allem mit Blick auf die zentrale Ausführungskomponente einer Workflow Engine, den Navigator, beschrieben und zusammengefasst. Die durch diese Betrachtung und Analyse gewonnenen Erkenntnisse werden anschließend in eine allgemeine Beschreibung der Funktionsweise eines Navigators umgesetzt. Dabei werden die grundlegenden Komponenten einer Workflow Engine identifiziert sowie ihre Funktionalität und ihr Zusammenspiel mit dem Navigator beschrieben. Der so hergeleitete modulare Aufbau einer Workflow Engine wird abschließend in zwei unterschiedlichen Anwendungsszenarien weiter vertieft. Es wird gezeigt, wie die beschriebene Modularität in bestimmten Szenarien als Vorteil genutzt werden kann.

Die Beschreibung der einzelnen Komponenten einer Workflow Engine ist in dieser Arbeit sehr abstrakt gehalten. Die Funktion der Komponenten wird allgemein beschrieben und an Beispielen erläutert. In späteren Arbeiten kann dieses näher ausspezifiziert werden. Die Schnittstellen der Komponenten können detaillierter ausgearbeitet und die Interaktion der einzelnen Komponenten näher untersucht werden.

Kapitel 6 führt die in Kapitel 4 erarbeiteten Verfahren zum Erzeugen von Wörtern aus Workflow Grammatiken mit dem in Kapitel 5 entwickelten modularen Aufbaue einer Workflow Engine zusammen. Es wird der Entwurf einer modularen Workflow Engine (Kapitel 5) für die Ausführung von Workflow Grammatiken (Kapitel 4) beschrieben. Die in Kapitel 3 vorgestellten Workflow Grammatiken werden für die Ausführung durch eine Workflow Engine in eine interne Repräsentation überführt. Ein grundlegender Aspekt dieser Überführung ist der Umbau der im Allgemeinen kontextsensitiven Workflow Grammatiken in kontextfreie Grammatiken. Dies wird dadurch erreicht, dass Nichtterminale aus den Produktionsregeln entfernt werden. Im Gegenzug dazu werden die so entstandenen Regeln mit zusätzlichen Metadaten angereichert. Neben der Herstellung der Kontextfreiheit werden die Regeln einer Workflow Grammatik bereits vor der Ausführung interpretiert und so organisiert, dass sie anschließend effizient ausgeführt werden können.

Der in [Vuk11] vorgestellte Entwurf von Workflow Grammatiken war zum Zeitpunkt der Erstellung dieser Arbeit noch nicht vollständig oder beendet. Die vorgestellte Überführung von Workflow Grammatiken in eine für die Ausführung optimierte interne Repräsentation kann in weiteren Arbeiten an die aktuelle Spezifikation von Workflow Grammatiken angepasst werden.

In Kapitel 7 wird die Implementierung eines Prototyps einer Workflow Engine für die Ausführung von Workflow Grammatiken skizziert. Die Implementierung folgt im Wesentlichen dem in Kapitel 6 entwickelten Entwurf. Es werden einige Details der Implementierung, wie beispielsweise die asynchrone Ausführung von lang laufenden Aktivitäten oder der Ruhezustand des Navigators vorgestellt und beschrieben.

Die in dieser Arbeit vorgestellte Implementierung ist lediglich ein einfacher Prototyp zur Demonstration des vorgestellten Entwurfs. Workflow Grammatiken übernehmen verschiedenen Konzepte in Bezug auf Datenmodell und Web Service Kommunikation von BPEL. Eine Weiterentwicklung des vorgestellten Prototypen kann darin bestehen, die Komponenten zur Datenhaltung und zur Web Service Kommunikation aus bereits bestehenden Workflow Engines für BPEL zu entnehmen und diese bereits vorhandenen Funktionalitäten in eine Workflow Engine zur Ausführung von Workflow Grammatiken zu integrieren.

9 Verzeichnisse

Literaturverzeichnis

AAD+04 Aalst, Wil M. P.; Aldred, Lachlan; Dumas, Marlon; Hofstede, Arthur H. M. (2004): Design and Implementation of the YAWL System. In: Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor et al. (Hg.): Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 142–159.

AH05 van der Aalst, W.M.P.; ter Hofstede, A.H.M. (2005): YAWL: Yet Another Workflow Language. In: Information Systems 30 (4), S. 245–275.

Bal09 Balzert, Helmut (2009): Lehrbuch der Softwaretechnik. Basiskonzepte und Requirements-Engineering. 3. Aufl. Heidelberg: Spektrum, Akad. Verl.

BKN11 Baun, Christian; Kunze, Marcel; Nimis, Jens; Tai, Stefan (2011): Informatik im Fokus - Cloud Computing. Web-basierte dynamische IT-Services. Berlin, Heidelberg: Springer Berlin Heidelberg.

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

EMW+05 Emig, Christian; Momm, Christof; Weisser, Jochen; Abeck, Sebastian (2005): Programming in the Large based on the Business Process Modelling Notation. In: Armin B. Cremers (Hg.): Informatik 2005 - Informatik LIVE! Beiträge der 35. Jahrestagung der Gesellschaft für Informatik (GI), 19. bis 22. September 2005 in Bonn. Bonn: Gesellschaft für Informatik, S. 627–631.

GHS95 Georgakopoulos, Diimitrios; Hornick, Mark; Sheth, Amit (1995): An overview of workflow management: From process modeling to workflow automation infrastructure. In: Distrib Parallel Databases 3 (2), S. 119–153.

HMG11 Heinisch, Cornelia; Müller-Hofmann, Frank; Goll, Joachim (2011): Java als erste Programmiersprache. Vom Einsteiger zum Profi. 6., überarb. Wiesbaden: Vieweg + Teubner.

Hoe10 Höing, André (2010): Orchestrating Secure Workflows for Cloud and Grid Services. Dissertation. TU Berlin, Berlin. Institut für Telekommunikationssysteme.
Online verfügbar unter http://opus.kobv.de/tuberlin/volltexte/2010/2817/.

Hol95 Hollingsworth, David (1995): Workflow Management Coalition - The Workflow Reference Model. Hg. v. Workflow Management Coalition (WfMC). Workflow Management Coalition.
Online verfügbar unter http://www.wfmc.org/reference-model.html.

JMS06 Juric, Matjaz B.; Mathew, Benny; Sarang, P. G. (2006): Business process execution language for web services. An architect and developer's guide to orchestrating web services using BPEL4WS, second edition. 2nd. Hg. v. Ashutosh Pande. Birmingham, U.K: Packt Pub.

LKN+09 Lenk, Alexander; Klems, Markus; Nimis, Jens; Tai, Stefan; Sandholm, Thomas (2009): What's inside the Cloud? An architectural map of the Cloud landscape. In: 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing. (CLOUD 2009) : May 23, 2009, Vancouver, Canada : in conjunction with the 2009 IEEE 31st International Conference on Software Engineering (ICSE 2009), May 16-24, 2009. International Conference on Software Engineering. Piscataway, N.J.: IEEE, S. 23–31.

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

MG11 Mell, Peter; Grance, Timothy (2011): The NIST Definition of Cloud Computing (Draft). Recommendations of the National Institute of Standards and Technology. National Institute of Standards and Technology, U.S. Department of Commerce.
Online verfügbar unter http://www.nist.gov/itl/cloud/.

Mue09 Müller, Horst M. (2009): Arbeitsbuch Linguistik. Eine Einführung in die Sprachwissenschaft. 2., überarb. u. aktualis. Stuttgart: UTB GmbH.

PCW85 Parnas, D.L; Clements, P.C; Weiss, D.M (1985): The Modular Structure of Complex Systems. In: IIEEE Trans. Software Eng SE-11 (3), S. 259–266.

Prz11 Przybylski, Diana (2011): Planungsverfahren im scientific Workflow Management. Diplomarbeit. Universität Stuttgart, Stuttgart. IAAS.
Online verfügbar unter http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-3058&mod=0&engl=0&inst=IAAS.

PW08 Priese, Lutz; Wimmel, Harro (2008): Petri-Netze. 2. Aufl. Berlin ;, Heidelberg: Springer.

Sch08 Schöning, Uwe (2008): Theoretische Informatik - kurz gefasst. 5. Aufl. Heidelberg: Spektrum, Akademischer Verlag.

Ste10 Steinmetz, Thomas (2008): Ein Event-Modell für WS-BPEL 2.0 und dessen Realisierung in Apache ODE. Diplomarbeit. Universität Stuttgart, Stuttgart. Institut für Architektur von Anwendungssystemen.
Online verfügbar unter http://elib.uni-stuttgart.de/opus/volltexte/2008/3617/.

Stu10 Stürmer, Gerhard (2010): An architectur of a workflow execution engine to enable network based execution of dynamic workflows. Masterarbeit. Universität Wien, Wien.
Online verfügbar unter http://othes.univie.ac.at/8672/.

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

Tho10 Thomas, Wolfgang (2010): ,,When nobody else dreamed of these things“ – Axel Thue und die Termersetzung. In: Informatik Spektrum 33 (5), S. 504–508.

Vuk11 Vukojevic, Karolina (2011): Transformation von BPEL Prozessmodellen in Grammatikbasierte Prozessmodelle. Diplomarbeit. Universität Stuttgart, Stuttgart. Institut für Architektur von Anwendungssystemen.

WCL+05 Weerawarana, Sanjiva; Curbera, Francisco; Leymann, Frank; Ferguson, Donald F.; Storey, Tony (2005): Web services platform architecture. SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and more. Upper Saddle River, NJ: Prentice Hall PTR.

WFMC99 Workflow Management Coalition (Hg.) (1999): Workflow Management Coalition - Terminology & Glossary. Workflow Management Coalition.
Online verfügbar unter http://www.wfmc.org/Glossaries-FAQs/View-category.html.

WH09 Wagenknecht, Christian; Hielscher, Michael (2009): Formale Sprachen, abstrakte Automaten und Compiler. Lehr- und Arbeitsbuch für Grundstudium und Fortbildung. Wiesbaden: Vieweg+Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden.

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

WSFL01 Snell, James (2001): The web services insider, Part 4: Introducing the Web Services Flow Language. IBM.
Online verfügbar unter http://www.ibm.com/developerworks/library/ws-ref4/.

Alle in diesem Dokument aufgeführten Weblinks wurden das letzte Mal am 28.08.2011 geprüft.

Abbildungsverzeichnis

Abbildung 1: Cloud Computing 8
Abbildung 2: Anwendung von Produktionsregeln 10
Abbildung 3: Grammatik, Ableitung und Ableitungsbaum 11
Abbildung 4: Chomsky Hierarchie 12
Abbildung 5: Endlicher Automat, nichtdeterministisch (NEA) 15
Abbildung 6: Endlicher Automat, deterministisch (DEA) 15
Abbildung 7: (Nichtdeterministischer) Kellerautomat 17
Abbildung 8: Einheitliche Ausführung von Prozessmodellen 21
Abbildung 9: Akzeptor und Generator 28
Abbildung 10: Repräsentationen von Prozessinstanzen 29
Abbildung 11: Regelanwendung regulärer Grammatiken 32
Abbildung 12: Gegenüberstellung Akzeptor und Generator für reguläre Sprachen 32
Abbildung 13: Kellerautomat als Generator 34
Abbildung 14: Worterzeugung, Beispiel 34
Abbildung 15: Verwaltung der Nichtterminale, kontextfreie Grammatik 37
Abbildung 16: Regelanwendung in kontextsensitiven Grammatiken 38
Abbildung 17: Workflow Management Coalition Reference Model 41

Abbildung 18: Apache ODE 44
Abbildung 19: Workflow Execution Engine (WEE) 45
Abbildung 20: Oracle BPEL Process Manager 46
Abbildung 21: YAWL Workflow System 47
Abbildung 22: Verwendung Prozessmodellspeicher 50
Abbildung 23: Prozessdatenspeicher 51
Abbildung 24: Prozessausführungszustand 51
Abbildung 25: Bedingter Kontrollfluss 52
Abbildung 26: Auswertung von Ausdrücken, Szenario 1 53
Abbildung 27: Auswertung von Ausdrücken, Szenario 2 54
Abbildung 28: Ausführung einer Aktivität mit Workitems 55
Abbildung 29: Ausführung einer Aktivität durch Programmaufruf 56
Abbildung 30: Aktivierung des Navigators 57
Abbildung 31: Komponenten einer Cloud Workflow Engine nach [Hoe10] 58
Abbildung 32: Workflow Engine als modulares System 59
Abbildung 33: Navigator als Workflow Engine + Dienste 60
Abbildung 34: Anwendungsszenario 1 61
Abbildung 35: Ausgangssituation 62
Abbildung 36: Installation und Ausführung 63
Abbildung 37: Anwendungsszenario 2 63
Abbildung 38: Variablenzugriff 65
Abbildung 39: Mobile Platzhalter 66
Abbildung 40: Aktivierungs- und Beendigungsregel 67
Abbildung 41: Beendigungsregeln mit Fehler- Nichtterminalen 68
Abbildung 42: Auswertung boolescher Bedingungen 68
Abbildung 43: Empfang einer Nachricht, bedingter Kontrollfluss 69
Abbildung 44: Erzeugen von Scopes 70
Abbildung 45: Erzeugen von mobilen Platzhaltern 70
Abbildung 46: Mobile Platzhalter als globale Zähler 71
Abbildung 47: Sequenzdiagramm Regelauswahl 77
Abbildung 48: Sequenzdiagramm synchrone Aktivität 78
Abbildung 49: Sequenzdiagramm asynchrone Aktivität 79
Abbildung 50: Überblick über die gesamte Arbeit 81

Tabellenverzeichnis

Tabelle 1: Nichtterminaltypen von Workflow Grammatiken 22

Listingverzeichnis

Listing 1: Beispiel für eine rechtsreguläre Grammatik 13
Listing 2: Beispiel für eine linksreguläre Grammatik 13
Listing 3: Beispiel für einen regulären Ausdruck 14
Listing 4: Beispiel für eine kontextfreie Grammatik 16
Listing 5: Beispiel für eine kontextsensitive Grammatik 18
Listing 6: Workflow Grammatik XML Darstellung 76

Definitionsverzeichnis

Definition 1: Formale Grammatik 9
Definition 2: Reguläre Ausdrücke 14
Definition 3: Endlicher Automat 14
Definition 4: (Nichtdeterministischer) Kellerautomat 16
Definition 5: Turingmaschine 19
Definition 6: Churchsche These (nach [Sch08]) 19

Erklärung

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

Unterschrift:

Stuttgart, 31.08.2011