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

Diplomarbeit Nr. 2523

Konzeption und Entwicklung einer AOP-fähigen BPEL Engine und eines Aspect-Weavers für BPEL Prozesse

Ralf Schroth

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. F. Leymann
Betreuer:	M.Sc. D. Karastoyanova
begonnen am:	24. Mai 2006
beendet am:	23. November 2006
CR-Klassifikation:	H.4.1, D.2.12, D.3.2, D.3.3, C.2.4, D.1.0

Inhaltsverzeichnis

Abbildungsverzeichnis vii

Tabellenverzeichnis ix

Listings xi

Abkürzungsverzeichnis xiii

1 Einleitung 1
- 1.1 Hintergrund . . . 1
- 1.2 Motivation . . . 2
- 1.3 Aufgabenstellung . . . 2
- 1.4 Gliederung der Arbeit . . . 3

2 Grundlagen 5
- 2.1 BPEL und flexible WS-Flows . . . 5
  - 2.1.1 Service Orientierte Architekturen und Web-Services . . . 5
  - 2.1.2 Workflows . . . 7
  - 2.1.3 WS-Flows und Orchestrierung von Web-Services mit WS-BPEL . . . 8
  - 2.1.4 Flexible Workflows und WS-Flows . . . 10
- 2.2 Aspekt-Orientierte Programmierung . . . 16
  - 2.2.1 Modularisierung von Softwaresystemen . . . 16
  - 2.2.2 Grundidee der aspektorientierten Programmierung . . . 16
  - 2.2.3 Bestandteile einer aspektorientierten Programmiersprache . . . 17
  - 2.2.4 Weaving-Varianten . . . 18
  - 2.2.5 Das Programmiermodell von AspectJ . . . 20
- 2.3 WS-Notification . . . 22
  - 2.3.1 Grundlagen von Publish/Subscribe Systemen . . . 22

2.3.2 Die WS-Notification Protokollfamilie . . . 24
2.3.3 WS-Eventing . . . 30
2.4 WS-Policy . . . 30
2.4.1 Web Services Policy Framework . . . 31
2.4.2 Web Services Policy Attachment . . . 34

3 Anforderungen und Konzeption 35
3.1 Anforderungen . . . 35
3.1.1 Anforderungen an Definition und Verwaltung der Aspekte . . . 35
3.1.2 Anforderungen an den Aspect-Weaver . . . 36
3.2 Konzeption . . . 37
3.2.1 Vorüberlegungen . . . 38
3.2.2 Systemarchitektur . . . 39
3.2.3 Anbindung an die Engine . . . 40
3.2.4 Konzeption des Weavers . . . 51
3.2.5 Konzeption des WS-Notification Wrappers . . . 67
3.2.6 Konzeption des Aspekt Management Tools . . . 68

4 Umsetzung 73
4.1 Eingesetzte Technologien . . . 73
4.1.1 Java . . . 73
4.1.2 BPEL-Engine . . . 73
4.1.3 Spring Framework 1.2 . . . 74
4.1.4 WS-Messenger . . . 75
4.1.5 Apache Neethi . . . 75
4.1.6 Apache Axis 1.4 . . . 76
4.1.7 AspectJ . . . 76
4.1.8 Java Message Service . . . 76
4.1.9 Apache Tomcat . . . 77
4.1.10 DOM4J . . . 77
4.2 Überblick . . . 77
4.3 Datenmodell . . . 79
4.3.1 Domänenmodell . . . 79
4.3.2 Serialisierung / Deserialisierung von Aspekten . . . 79
4.4 BPEL Event Framework und Controller . . . 80
4.4.1 Integration in die XEngine . . . 80
4.4.2 Realisierung des Controller-Frameworks . . . 83
4.5 Aspekt Weaver . . . 86
4.5.1 Registries . . . 87
4.5.2 AspectManager . . . 89
4.5.3 AspectMatcher . . . 91
4.5.4 AspectDeployer . . . 91
4.5.5 Deployment Web-Service . . . 92

4.5.6 AdviceScheduler . . . 92
4.5.7 AdviceInvoker . . . 93
4.5.8 Notification Adapter . . . 93
4.5.9 Marshaller . . . 95
4.5.10 EngineAdapter . . . 96
4.6 Aspekt Management Tool . . . 96
4.6.1 Präsentationsschicht . . . 97
4.6.2 Business-Logik-Schicht . . . 97
4.6.3 Persistenz-Schicht . . . 98
4.7 WS-Notification Wrapper . . . 99
4.7.1 Notification Consumer . . . 99
4.7.2 Wrapper . . . 99
4.7.3 Invoker . . . 100
4.7.4 Notification Producer . . . 100
4.8 Beispielprozess . . . 100
4.8.1 Szenario: Weihnachtsrabatt . . . 102
4.8.2 Szenario: Einführung der Bonitätsprüfung . . . 104
4.8.3 Szenario: Benachrichtigung der Versandabteilung . . . 105
4.8.4 Szenario: Dank-eMail an Kunden . . . 106
4.8.5 Szenario: Neuer Logistikpartner . . . 108

5 Zusammenfassung und Ausblick 111
5.1 Zusammenfassung . . . 111
5.2 Ausblick . . . 112

A Schema-Definitionen der Policy-Assertions 115

B Schema-Definitionen der Notification Nachrichten 119

C Schema-Definitionen der Event Deployment Deskriptoren 121

D Beispielprozess 123

Literaturverzeichnis 135

Abbildungsverzeichnis

2.1 Rollen in der Service Orientierten Architektur 6
2.2 Grundstruktur eines BPEL-Prozesses 9
2.3 Invasivität verschiedener Weaving-Mechanismen 19
2.4 Überblick über WS-BaseNotification 26
2.5 Überblick über WS-BrokeredNotification 29
2.6 Überblick über WS-Policy 32

3.1 Struktur eines Aspekts 36
3.2 Konzeptionelle Architektur des Systems 39
3.3 Zustandsdiagramm für allgemeine Aktivitäten 42
3.4 Zustandsdiagramm für Scopes 44
3.5 Zustandsdiagramm für Loops 45
3.6 Zustandsdiagramme für Prozesse und Links 46
3.7 Architektur des Event Frameworks 50
3.8 Systemarchitektur mit Event-Publishing-System 51
3.9 Struktur der Kommunikation zwischen Weaver und Wrapper 64
3.10 Use-Case Diagramm für das Aspect Management Tool 68
3.11 Hauptfenster des Aspekt Management Tools 69
3.12 Fenster für das Deployment einer Aspektgruppe 69
3.13 Fenster zum Anlegen einer neuen Aspektgruppe 70
3.14 Fenster zum Anlegen eines Aspekts 70
3.15 Fenster zum Definieren eines Advice 70
3.16 Fenster zum Anlegen eines Pointcut 71

4.1 Laufzeitarchitektur des Systems 78
4.2 Domänenmodell für die AOP-fähige BPEL Engine 79
4.3 Klassendiagramm des Serialisierungsmechanismus 80

4.4 Integration des Event Frameworks in die XEngine 81
4.5 Hierarchie der Eventmessages 82
4.6 Kommunikation des Event Frameworks 83
4.7 Schichtenarchitektur des Controller-Frameworks (grau hinterlegte Komponenten sind anwendungsspezifisch) 84
4.8 Entwurf des Controller Frameworks 85
4.9 Schichtenarchitektur des Weavers 86
4.10 Interne Architektur des Weavers 87
4.11 Ablauf des Weavings 88
4.12 Struktur der Persistenz für Aspektgruppen 89
4.13 Aufbau der vom AspectManager verwendeten Datenstruktur 90
4.14 Interaktion von Weaver, AspectMatcher und AspectManager 91
4.15 Struktur für das Deployment von Aspekten 92
4.16 Struktur des Aufrufs von Advices 94
4.17 Datenstruktur für die Ablage der Resultate eines Web-Service Aufrufs 95
4.18 Schichtenarchitektur des Aspect Management Tools 97
4.19 Architektur des WS-Notification Wrappers 99
4.20 Struktur des Beispielprozesses 101

Tabellenverzeichnis

3.1 Liste der Events 49
3.2 Abbildung der Joinpoint- und Advice-Typen auf Events 53
3.3 Regeln für Bestimmung der Parameterwerte 60
3.4 Beschreibung des Deployment Web-Service 66

Listings

2.1 Beispiel für eine WS-Policy . . . 33
3.1 Struktur einer Aspekt-Definition . . . 54
3.2 Struktur eines Policy-Attachments für Aspekte . . . 56
3.3 Beispiel eines Aspekts inkl. Attachment . . . 57
3.4 Struktur einer Invocation Notification Message . . . 64
3.5 Struktur einer Response Notification Nachricht . . . 65
3.6 Beispiel für eine vom Aspekt Management Tool genutzte Endpunktreferenz . . . 71
4.1 Beispiel für einen Event Deployment Descriptor . . . 83
4.2 Ausschnitt aus der Definition des Aspekts für die Rabattaktion . . . 102
4.3 Ausschnitt aus der Definition des Aspekts für die Bonitätsprüfung . . . 104
4.4 Ausschnitt aus der Definition des Aspekts für die Benachrichtigung bei Kompensation 106
4.5 Definition des Aspekts für den Versand von Dank-eMails an wichtige Kunden . . . 107
4.6 Ausschnitt aus der Definition des Aspekts für die Nutzung eines neuen Logistikpartners 108
A.1 Schema-Definitionen der Policy-Assertions . . . 115
B.1 Schema der Notification Nachrichten . . . 119
C.1 Schema der Event Deployment Deskriptoren . . . 121
D.1 BPEL-Beschreibung des Beispielprozesses . . . 123
D.2 WSDL-Definitionen der im Beispielprozess verwendeten Web-Services . . . 126

Abkürzungsverzeichnis

AOP	**A**spekt**o**rientierte **P**rogrammierung
BPEL	**B**usiness **P**rocess **E**xecution **L**anguage for Web-Services
BPR	**B**usiness **P**rocess A**r**chive
DOM	**D**ocument **O**bject **M**odel
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
IoC	**I**nversion **o**f **C**ontrol
MOM	**M**essage **O**riented **M**iddleware
OO	**O**bjekt**o**rientiert
OOP	**O**bjekt**o**rientierte **P**rogrammierung
PIID	**P**rocess **I**nstance **Id**entifier
POJO	**P**lain **O**ld **J**ava **O**bject
QName	**Q**ualified **Name**
RPC	**R**emote **P**rocedure **C**all
SOA	**S**ervice-**O**rientierte **A**rchitektur
UDDI	**U**niversal **D**escription **D**iscovery and **I**ntegration
WfMS	**W**ork**f**low **M**anagement **S**ystem
WS-BPEL	siehe BPEL
WSDL	**W**eb **S**ervice **D**escription **L**anguage
XML	E**x**tensible **M**arkup **L**anguage

KAPITEL 1

Einleitung

1.1 Hintergrund

In der heutigen Zeit sehen sich Unternehmen immer größerem Wettbewerbsdruck ausgesetzt. Einer der Kernpunkte, die für den Erfolg oder Misserfolg eines Unternehmens entscheidend sind, sind die Geschäftsprozesse innerhalb des Unternehmens und zunehmend auch die unternehmensübergreifenden Prozesse entlang der Wertschöpfungskette. Insbesondere für Dienstleistungsunternehmen, bei denen der Geschäftsprozess und dessen Ausführung das eigentliche Produkt darstellt, ist dies von Bedeutung. Zur Optimierung der Prozesse wird versucht, diese auf Basis formalisierter Beschreibungen weitgehend automatisiert als sog. Workflows elektronisch auszuführen.

Durch den zunehmenden Trend zum Outsourcing und die Konzentration auf das Kerngeschäft wird es zunehmend notwendig, dass Unternehmen sich mit externen Partnern integrieren und ihre Geschäftsprozesse an die sich ändernden Bedingungen anpassen. Um Flexibilität zu gewinnen und die Interoperabilität zwischen verschiedenen Applikationen und Partnern zu verbessern, ist die serviceorientierte Architektur (SOA) ein häufig gewähltes Mittel. Web-Services sind hierbei momentan die populärste Technologie zur Realisierung einer SOA. Ein sehr bedeutender Aspekt der SOA ist die Komposition von einzelnen Diensten zu Prozessen. Auf der Seite der Web-Service Technologie spricht man in diesem Zusammenhang von Web-Service Flows (kurz WS-Flows), einer speziellen Art der Workflows. Eine weit verbreitete Möglichkeit zur Realisierung von WS-Flows ist die Business Process Execution Language (BPEL).

1.2 Motivation

Die durch BPEL-Prozesse abgebildeten Web-Service Flows werden zur Modellierungszeit statisch beschrieben. Die verwendeten Web-Services werden bei der Modellierung über ihren Porttyp spezifiziert. Ebenso wird die Struktur des Kontrollflusses während der Modellierung festgelegt. Beides schränkt die Flexibilität solcher Prozesse stark ein.

Um in der Lage zu sein, sich auf sich ändernde Anforderungen einzustellen, ist es für Unternehmen wichtig, ihre Geschäftsprozesse flexibel anpassen zu können. Gerade im sehr dynamischen Web-Service Umfeld ist dies von großer Bedeutung, da hier oft neue Partner auftreten, bestehende verschwinden oder Änderungen notwendig werden.

Es gibt bereits Ansätze, eine flexible Adaption von BPEL-Prozessen zu erlauben. Diese beschränken sich jedoch häufig auf eine dynamische Selektion der genutzten Web-Services zur Laufzeit. Es sind jedoch mächtigere Mechanismen erforderlich, um auch hinsichtlich der Struktur der Prozesse die Möglichkeit der flexiblen Adaption zu erreichen.

Das Paradigma der aspektorientierten Programmierung hat sich in den letzten Jahren als einfach anzuwendender Mechanismus zur Adaption und flexiblen Modularisierung von konventionellen Softwaresystemen erwiesen. Es liegt nahe, dieses Paradigma auf die Welt der WS-Flows zu übertragen und zu untersuchen, ob hierdurch eine größere Flexibilität bei der Nutzung von elektronischen Geschäftsprozessen erreicht werden kann.

1.3 Aufgabenstellung

In dieser Diplomarbeit soll ein Vorgehen entworfen und umgesetzt werden, mit dem eine BPEL-Engine um Fähigkeiten zur aspektorientierten Programmierung erweitert werden kann. Hierbei sollen insbesondere etablierte Standards aus dem Web-Services-Bereich, wie z. B. WS-Policy und WS-Notification zum Einsatz kommen.

Die Aufgabenstellung umfasst die Ist-Analyse der bestehenden erweiterten BPEL-Engine, den Entwurf und die Realisierung eines Werkzeugs zur Spezifikation- und Verwaltung von Aspekten, sowie den Entwurf und die Realisierung eines dynamischen Aspekt-Weavers, der die Aspekte instanzbasiert und zur Laufzeit in den BPEL-Prozess einwebt. Gegenstand der Einwebung sollen allgemeine Web-Service Aufrufe sein. Die Erweiterungen sollen möglichst wenige Änderungen an der bestehenden BPEL-Engine erfordern, um die Zusammenarbeit mit anderen, teilweise schon existierenden Erweiterungen und auch BPEL-Prozessen sicherzustellen.

Die Tauglichkeit des gewählten Ansatzes soll anhand von Beispielprozessen gezeigt werden.

1.4 Gliederung der Arbeit

Der Hauptteil dieser Arbeit ist wie folgt gegliedert:

Kapitel 2 – Grundlagen beschreibt die Grundlagen dieser Arbeit. Zunächst wird auf Workflows und insbesondere BPEL-Prozesse eingegangen und Konzepte zu ihrer Flexibilisierung vorgestellt. Daran schließt sich eine Betrachtung des Konzepts der aspektorientierten Programmierung und ihrer Eigenschaften an. Im Anschluss daran wird WS-Notification als interoperabler Standard für Publish/Subscribe Systeme auf Basis von Web-Services vorgestellt und zuletzt wird WS-Policy als Standard zur Beschreibung der über die Schnittstelle hinausgehenden Eigenschaften eines Web-Service beschrieben.

Kapitel 3 – Anforderung und Konzeption stellt zunächst die Anforderungen an eine aspektorientierte BPEL-Engine dar. Im Anschluss daran wird ein Konzept für eine AOP-fähige BPEL-Engine und einen Aspekt-Weaver für BPEL-Prozesse ausführlich beschrieben.

Kapitel 4 – Umsetzung beschreibt, wie das im vorhergehenden Kapitel beschriebene Konzept für eine AOP-fähige BPEL-Engine in eine prototypische Implementierung umgesetzt wurde und beschreibt deren Realisierung näher. Im Anschluss daran wird anhand eines Beispielprozesses und mehrerer Szenarien die Tauglichkeit des Ansatzes untersucht.

Kapitel 5 – Zusammenfassung und Ausblick fasst die Ergebnisse dieser Arbeit nochmals zusammen und beschreibt Punkte, die in dieser Arbeit nicht näher betrachtet wurden, jedoch interessant für die weitere Entwicklung sein könnten.

KAPITEL 2

Grundlagen

Dieses Kapitel gibt einen Überblick über die Grundlagen dieser Arbeit und vermittelt die für das Verständnis der folgenden Kapitel notwendigen Grundlagen. Hierzu werden die Grundlagen von BPEL und flexiblen Web-Service Flows dargestellt. Anschließend wird auf das Konzept der aspektorientierten Programmierung eingegangen. Im Anschluss daran werden Publish/Subscribe Systeme und deren Entsprechung im Web-Service Umfeld betrachtet. Abschließend wird WS-Policy als Mechanismus zum Beschreiben von Charakteristiken und Anforderungen von Web-Services vorgestellt.

2.1 BPEL und flexible WS-Flows

2.1.1 Service Orientierte Architekturen und Web-Services

Nach wie vor ist die Kooperation zwischen Unternehmensbereichen und ganzen Unternehmen von größter Bedeutung. Entwicklungen wie Outsourcing und „Virtuelle Unternehmen“ (vgl. [Leymann und Roller (2000)]) sind Beispiele für diese Tatsache. In diesem Zusammenhang sind die Begriffe des *Service* und der *Service Orientierten Architektur*, kurz SOA, (vgl. [Natis (2003); Weerawarana u. a. (2005)]) von zentraler Bedeutung. Ein Service beschreibt in diesem Zusammenhang ein Softwaremodul, auf das über einen Namen und eine definierte Schnittstelle zugegriffen wird. Die Service Orientierte Architektur ist eine neue Softwarearchitektur, die mit der Definition der Schnittstellen beginnt und die gesamte Topologie einer Anwendung als Topologie von Schnittstellen, Schnittstellen-Implementierungen und Aufrufen von Schnittstellen definiert (vgl. [Natis (2003)]). Hierbei können die Begriffe Service und Schnittstelle als gleichbedeutend angesehen werden. Ähnliche Definition der SOA und des Service-Begriffs finden sich beispielsweise auch bei [Burbeck (2000)], [MacKenzie u. a. (2006)] sowie [Weerawarana u. a. (2005)]. Ein wesentliches Kennzeichen einer SOA ist die *Lose Kopplung* (vgl. [Kaye (2003)]) der Services.

Charakteristisch für eine SOA ist das sog. *SOA Dreieck*, das in Abbildung 2.1 abgebildet ist. Es beschreibt die Interaktion zwischen Dienstanbietern (*Service Provider*), Dienstnutzern (*Service Consumer*) und dem Dienstvermittler (*Service Registry*), der anhand einer Schnittstellenbeschreibung und weiteren Kriterien einem Dienstnutzer einen passenden Dienstanbieter vermittelt. Hierzu registriert ein Dienstanbieter die von ihm bereitgestellten Services und optional nähere Beschreibungen derer Eigenschaften und Nutzungsbedingungen beim Dienstvermittler in einem *publish* genannten Schritt. Ein Dienstnutzer, der auf der Suche nach einem Dienstanbieter ist, tritt an den Dienstvermittler mit einer Beschreibung des gesuchten Service, und evtl. weiteren Kriterien heran und erhält eine Liste der in Frage kommenden Dienstanbieter zurück. Dieser Schritt wird als *find* Schritt bezeichnet. Im nun folgenden *bind* Schritt, sucht sich der Dienstnutzer aus den Ergebnissen einen Dienstanbieter heraus, bindet sich an diesen und ruft die gewünschten Funktionen auf.

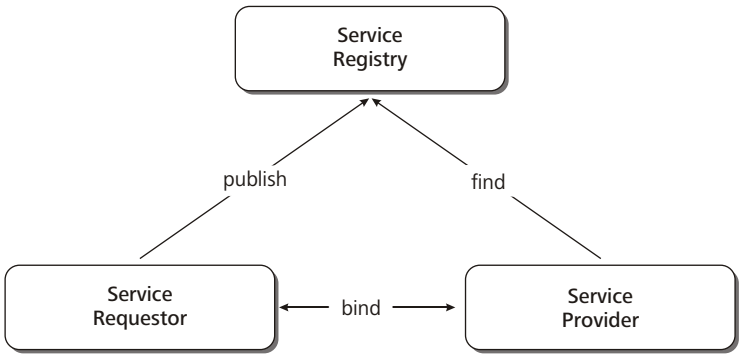


Abbildung 2.1: Rollen in der Service Orientierten Architektur

Web-Services (vgl. [Booth u. a. (2004)]) sind eine Technologie, die häufig für die Realisierung einer SOA genutzt wird. [Gottschalk (2000)] definiert einen Web-Service wie folgt:

> *„Web Services are self-contained, modular applications that can be described, published, located, and invoked over a network, generally, the Web."*

Ein Web-Service ist also eine über ein Netzwerk verfügbar gemachte Funktionalität, die eine definierte Schnittstelle besitzt. Ein Web-Service erfüllt demnach die Definition einer Softwarekomponente nach [Szyperski (1998)]. Nach [Leymann (2003)] kann ein Web-Service als *virtuelle Komponente* angesehen werden, da weder das für die Implementierung des Web-Service genutzte Programmiermodell, noch das zugrundeliegende Komponentenmodell für die Nutzung der Komponente von Bedeutung sind.

Die gesamte Web-Service Plattform besteht aus einer Vielzahl von modularen und aufeinander aufbauenden Spezifikationen, die überwiegend als offene Industriestandards entstanden sind. Eines der wichtigsten Ziele bei der Einführung der Web-Service Plattform war die Interoperabilität. Hierdurch wird es möglich Web-Services in heterogenen Umgebungen zur Anwendungsintegration zu nutzen und für Legacy-Applikationen Adapter zu erstellen,

die die Legacy-Funktionalität virtualisieren und im Rahmen einer SOA nutzbar machen.

2.1.2 Workflows

In der heutigen Zeit gewinnen die Geschäftsprozesse, mit denen Unternehmen ihre Produkte erstellen und vertreiben immer größere Bedeutung. Insbesondere für Dienstleistungsunternehmen, bei denen der Geschäftsprozess das Produkt darstellt, ist dies zu beobachten. Für den Begriff Geschäftsprozess selbst gibt es diverse Definitionen, stellvertretend sei hier die Definition der [Workflow Management Coalition (1999)] angegeben:

> „*A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships.*“

Durch die zunehmende Intensität des wirtschaftlichen Wettbewerbs und die dadurch erwachsenden Anforderungen nach immer schnelleren Reaktionszeiten und Entwicklungszyklen wächst der Druck auf die Unternehmen ihre Geschäftsprozesse zu optimieren und effizienter zu gestalten. Im Zuge dessen wurde zu Beginn der 90er Jahre damit begonnen die Geschäftsprozesse als sog. *Workflows* zu formalisieren und in IT-Systeme abzubilden. Die [Workflow Management Coalition (1999)] definiert einen Workflow wie folgt:

> „*The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.*“

Zur Ausführung von Workflows auf IT-Infrastrukturen kommt ein sog. *Workflow Management System (WfMS)* zum Einsatz. Die [Workflow Management Coalition (1999)] gibt folgende Definition für ein WfMS an:

> „*A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.*“

Obige Definition beschreibt die gesamte Umgebung, die für die IT-gestützte Ausführung von Geschäftsprozessen notwendig ist. Zentraler Bestandteil und verantwortlich für die Ausführung von Workflows ist die sog. *Workflow Engine* (vgl. [Workflow Management Coalition (1999)]).

Die Modellierung von Workflows trennt sehr scharf zwischen den innerhalb eines Prozesses zu nutzenden Funktionen und dem Kontrollfluss, auch Kontrollprogramm genannt, der diese Funktionen verbindet. Das Kontrollprogramm bildet den Kern eines Workflows. Hierdurch wird die Freiheit gewonnen, den Ablauf des Workflows unabhängig von den darin genutzten Funktionen ändern zu können. Die Entwicklung von workflowbasierter Software gliedert sich dadurch in zwei Ebenen (vgl. [Leymann und Roller (1997)]):

Programming in the small Die Entwicklung der Funktionen, die für den Workflow bereitstehen. Die Funktionen werden i. d. R. als Komponenten mit definierten Schnittstellen entwickelt und können so zu komplexen Geschäftsprozessen zusammengesetzt werden.

Programming in the large Die Beschreibung des Kontroll- und Datenflusses zwischen den einzelnen Funktionen, der beteiligten Rollen und Ressourcen (auch als Prozesslogik-, Organisations- und Infrastruktur-Dimension eines Workflows (vgl. [Leymann und Roller (2000)]) bezeichnet). Hier wird die Komposition von Funktionen und Teil-Prozessen zu größeren Prozessen vorgenommen.

2.1.3 WS-Flows und Orchestrierung von Web-Services mit WS-BPEL

Da Web-Services weitgehend den Funktionen in den klassischen Workflows entsprechen, liegt es nahe, Web-Services auch als Bestandteil von Workflows zu verwenden. Bei der Beschreibung solcher Workflows sind zwei Perspektiven zu unterscheiden:

- Orchestration
- Choreography

Orchestration bezieht sich auf die Betrachtung eines Geschäftsprozesses aus der Perspektive eines einzelnen Partners, wohingegen Choreography das Gesamtbild der Interaktion zwischen den einzelnen Beteiligten beschreibt.

Eine Orchestrierung von Web-Services wird auch als *Web-Service Flow* oder kurz *WS-Flow* bezeichnet. WS-Flows unterscheiden sich von herkömmlichen Workflows dadurch, dass sie sich auf die Interaktion zwischen Web-Services beschränken und die organisatorische oder „wer“-Perspektive (d. h. welcher Mitarbeiter ist mit welchem Prozesschritt verbunden) ausklammern. Es gibt jedoch Forschungen mit dem Ziel auch menschliche Akteure in WS-Flows einzubinden, wie z. B. BPEL4People (vgl. [Kloppmann u. a. (2005)]).

Eine Sprache um Orchestrierungen von Web-Services zu beschreiben, die sich zum de facto Standard für diese Aufgabenstellung entwickelt hat, ist die *Business Process Execution Language for Web Services* (vgl. [Curbera u. a. (2003)]), kurz *BPEL* genannt. BPEL entstand aus den

beiden konkurrierenden Spezifikationen WSFL (vgl. [Leymann (2001)]) von IBM, die eine graphbasierte Modellierung der Prozesse erlaubte, und XLANG (vgl. [Thatte (2001)]) von Microsoft, das einen algebraischen Ansatz verfolgte. In BPEL sind die Modellierungsansätze seiner beiden Vorläufer vereinigt.

BPEL ist eine XML-basierte Sprache zur Beschreibung eines Geschäftsprozesses aus der Perspektive eines Teilnehmers. Hierzu werden bestehende Standards aus der Web-Service Welt wie WSDL verwendet. Die Komposition der Web-Services zu einem Prozess erfolgt abstrakt, da nur die Porttypen der genutzten Web-Services spezifiziert werden, jedoch keine konkreten Endpunkte angegeben sind. Diese werden erst beim *Deployment* des Prozesses festgelegt. Eine hierarchische Komposition von BPEL-Prozessen wird dadurch ermöglicht, dass die Orchestrierung selbst als ein neuer bzw. mehrere neue Web-Services verfügbar gemacht wird.

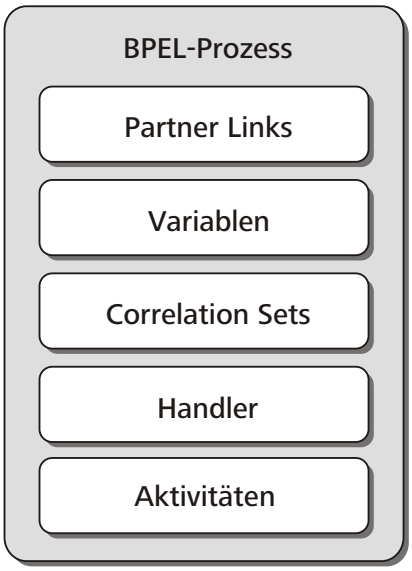


Abbildung 2.2: Grundstruktur eines BPEL-Prozesses

Der generelle Aufbau eines BPEL-Dokuments ist in Abbildung 2.2 dargestellt. Der Abschnitt „Partner Links“ modelliert bidirektionale Kommunikation und beschreibt die Porttypen, der am Prozess beteiligten Web-Services. Auf die Abschnitte zur Deklaration von Variablen, Correlation Sets und Handlern wird hier nicht näher eingegangen.

Die eigentliche Beschreibung des Ablaufs eines Geschäftsprozesses erfolgt im Abschnitt „Aktivitäten“. Eine solche Beschreibung besteht aus verschiedenen sog. *Aktivitäten*, die entweder *atomar* oder *strukturiert* sein können. Atomare Aktivitäten beschreiben die Interaktion des Geschäftsprozesses mit anderen Web-Services (z. B. Aufrufen eines WS, Empfangen eines Aufrufs etc.), die Manipulation von Daten und das Auslösen von Ereignisbehandlungen. Strukturierte Aktivitäten dienen dazu, den Kontrollfluss zu modellieren (z. B. Verzweigun-

gen, Schleifen, Parallelausführung von Aktivitäten etc.) Über die Aktivitäten hinaus gibt es noch sog. *Links*, die genutzt werden können, um explizit den Kontrollfluss zwischen zwei Aktivitäten zu modellieren. Links dürfen hierbei nur innerhalb einer Flow-Aktivität definiert werden. Solche Links können zusätzlich mit Prädikaten (sog. *Transition Conditions*) annotiert sein, um bedingte Kontrollflüsse zu modellieren.

Ausführliche Beschreibungen von BPEL findet sich bei [Weerawarana u. a. (2005)] und [Alonso u. a. (2004)].

2.1.4 Flexible Workflows und WS-Flows

2.1.4.1 Flexible Workflows

Wie schon in Abschnitt 2.1.2 beschrieben sind Unternehmen und Geschäftsprozesse zunehmend schnellerem Wandel unterworfen. Um diesen Anforderungen gerecht zu werden, bedarf es auch Mechanismen zur Anpassung von Workflows. Man spricht in diesem Zusammenhang auch von *flexiblen Workflows* (vgl. [Casati u. a. (1996)]). Mechanismen für flexible Workflows versuchen, die *Evolution* der Workflows möglichst gut zu unterstützen. Evolution von Workflows bedeutet in diesem Zusammenhang die Änderung des Schemas eines bestehenden Workflows. [Casati u. a. (1996)] unterscheiden zwei Arten von Workflow Evolution:

Statische Workflow Evolution Beschreibt die Modifikation des Workflow-Schemas, so dass ein Workflow angepasst werden kann, ohne ihn von Grund auf neu erstellen zu müssen

Dynamische Workflow Evolution Beschreibt die Modifikation der Workflowbeschreibung bereits laufender Instanzen eines Workflows oder die Möglichkeit, laufende Instanzen zu aktualisieren, um sie an ein modifiziertes Schema anzupassen.

Insbesondere die dynamische Workflow Evolution ist nicht trivial. Simple Ansätze wie den Prozess unverändert zu Ende laufen (sog. *Flush Policy*) zu lassen, können nicht tauglich sein, da beispielsweise Änderungen der gesetzlichen Rahmenbedingungen dies nicht zulassen würden, insbesondere bei langlaufenden Prozessen. Auch der Ansatz eine Instanz abzubrechen und zu kompensieren, um sie anschließend mit verändertem Schema erneut zu starten (sog. *Abort Policy*) ist nicht tauglich, da es oft nicht möglich ist, langlaufende Prozesse abzubrechen und neu zu starten. Ein komplexerer Ansatz (sog. *Progressive Policy*), der die verschiedenen Zustände, in denen sich ein Prozess bezüglich den Änderungen befinden kann, berücksichtigt und entsprechende Strategien hierfür vorsieht, ist erforderlich. Details dieses Ansatzes werden bei [Casati u. a. (1996)] beschrieben.

Evolutionäre Veränderungen von Workflows sind nicht die einzigen Situationen, in denen ein Workflow geändert werden muss. Ebenso besteht die Möglichkeit, dass Situationen (z. B. Benutzereingriff wird erforderlich, unvorhersehbare Ereignisse, Fehlerfälle etc.) eintreten, die bei der Modellierung des Workflows nicht beachtet wurden und *ad-hoc* Änderungen, also Änderungen an laufenden Instanzen eines Prozesses, erfordern (vgl. [Han u. a. (1998)]). Auch hierfür sollten in einem flexiblen Workflowsystem entsprechende Mechanismen existieren.

Es gibt mehrere Arten, wie man Flexibilität für Workflows realisieren kann (vgl. [Heinl u. a. (1999)]):

Flexibility by Selection Bei diesem Ansatz enthält die Workflowbeschreibung explizit oder implizit eine Reihe von vorhergesehenen Ausführungspfaden, aus denen der Benutzer den passenden auswählen kann. Alternative Pfade in Prozessen realisieren genau denselben Mechanismus, wobei hier keine Interaktion mit dem Benutzer erforderlich ist. Die alternativen Ausführungspfade müssen nicht zwingend alle bereits im voraus beschrieben sein, sondern können auch über einen Blackbox Mechanismus, d. h. genau definierte Stellen im Kontrollfluss, die als Platzhalter dienen und zur Laufzeit ersetzt werden, dynamisch hinzugefügt werden.

Flexibility by Adaptation Dieser Ansatz sieht vor, dass zur Laufzeit Ausführungspfade geändert oder hinzugefügt werden können. Hierbei können einzelnen Instanzen (*Instance Adaptation*) oder das Prozessmodell für zukünftige Instanzen (*Type Adaptation*) angepasst werden.

Flexibility by Selection erfordert, dass man bereits zum Zeitpunkt der Modellierung des Workflows die möglichen Änderungen, die in der Zukunft erforderlich werden können, kennt. Dies ist oft nicht der Fall, so dass Flexibility by Selection nicht für alle Anwendungsfälle ausreichend ist.

Flexibilität in Workflows ist in allen drei Workflow-Dimensionen (Prozesslogik-, Organisations- und Infrastruktur-Dimension) erforderlich. Hierbei muss jede Dimension separat gehandhabt werden und bedarf spezieller Adaptionsansätze.

Besonders die Flexibilität der Prozesslogik-Dimension ist von starkem Interesse. [Casati u. a. (1996)] beschreiben grundlegende Operationen und Regeln für ihre Anwendbarkeit, mit denen ein Workflow konsistent geändert werden kann. Die folgenden Operationen werden beschrieben:

- Hinzufügen einer neuen Aktivität nach einer bestehenden
- Entfernen einer Aktivität
- Ändern einer Aktivierungsbedingung

- Hinzufügen von Deklarationen (Variablen, Aktivitäten)
- Entfernen von Variablen-Deklarationen

Einen ähnlichen Ansatz zur dynamischen Änderung von laufenden Instanzen beschreiben [Reichert und Dadam (1998)] mit dem graphbasierten Workflow-Modell $\text{ADEPT}_{\text{flex}}$ und in dessen Weiterentwicklung ADEPT2 (vgl. [Reichert u. a. (2005)]). Auch hier werden Operationen zum dynamischen Einfügen, Löschen von Aktivitäten und zur Änderung der Reihenfolge von Aktivitäten sowie konsistenzerhaltende Regeln für deren Anwendung definiert.

Eine eingehende Diskussion der möglichen Fehler bei der Adaption von Workflows sowie eine Taxonomie der Gründe für Änderungen von Workflows findet sich bei [van der Aalst und Jablonski (2000)].

Die oben vorgestellten Ansätze basieren alle auf einer Transformation der Workflowbeschreibung. Die Modellierung von Geschäftsprozessen kann jedoch auch als eine Art zu programmieren („programming in the large“) angesehen werden, weshalb es sich anbietet, Paradigmen aus der Welt der Programmierung zu übertragen, um mehr Flexibilität für Workflows zu realisieren. Die aspektorientierte Programmierung beispielsweise bietet viele Möglichkeiten um dynamische Änderungen an Softwaresystemen vorzunehmen und bietet sich auch an um flexible Workflows zu realisieren (siehe Abschnitt 2.2).

[Schmidt und Assmann (1998)] skizzieren einen Ansatz zur Unterstützung von statischer Workflow Evolution von bestehenden objektorientierten Workflow-Implementierungen durch die Nutzung von aspektorientierter Programmierung (siehe Abschnitt 2.2). Sie setzen bei der von [Jablonski u. a. (1995)] beschriebenen Dekomposition eines Workflows in Aspekte (der Begriff hier deckt sich nicht vollständig mit der Definition eines Aspekts in der aspektorientierten Programmierung) und beschreiben darauf aufbauend, wie durch die Nutzung von Methoden der aspektorientierten Programmierung eine Adaption der Workflows ermöglicht wird. Ihr Ansatz basiert auf statischem Weaving (siehe Abschnitt 2.2.4) von Funktionen mit definierten Schnittstellen (bei Schmidt und Assmann „Komponenten“ genannt). In der Konsequenz bedeutet dies jedoch, dass keine Änderungen zur Laufzeit möglich sind.

[Bachmendo und Unland (2001)] beschreiben einen ähnlichen Ansatz, allerdings zur dynamischen Workflow Evolution. Der Ansatz nutzt dynamische AOP um die Adaption von objektorientierten Workflow-Implementierungen zur Laufzeit zu erlauben. Bachmendo und Unland verwenden Aspekte, um Kontrollfluss, Ressourcenzuweisung und Auditing zu adaptieren. Zur Adaption des Kontrollflusses werden beispielhaft „before“- und „instead“-Advices, sowie das Einfügen von And-Splits und -Joins sowie Iterationen gezeigt. Durch den aspektorientierten Ansatz werden sowohl die Workflows als auch die Adaptionen in Form der Aspekte wiederverwendbar.

Die beiden hier vorgestellten auf AOP basierenden Ansätze werden von [Courbis und Finkelstein (2004)] als nicht weitgehend genug bezeichnet, da sie die aspektorientierte Programmierung nur zum „Patchen“ von Prozessmodellen nutzen und nicht zur Modularisierung von Crosscutting-Concerns (siehe Abschnitt 2.2.1) einsetzen.

2.1.4.2 Flexible WS-Flows

Die in Abschnitt 2.1.4.1 beschriebenen Ansätze zur Flexibilität von Workflows können auch auf WS-Flows angewendet werden. Hierbei ist die in Abschnitt 2.1.3 beschriebene Einschränkung, dass WS-Flows die Organisations-Dimension von Workflows ignorieren, zu beachten.

Es gibt jedoch einen Unterschied hinsichtlich der Treiber für den Bedarf nach Adaption (vgl. [Erradi u. a. (2005)]). Die bei klassischen Workflows häufigen Treiber wie infrastrukturelle Änderungen sind für Web-Services mit ihren spezifizierten Schnittstellen irrelevant, wohingegen Web-Service Orchestrierungen viel häufiger mit Änderungen von Business-Rules und sich ändernden Policies konfrontiert sind.

Durch die konsequente Nutzung von Schnittstellen sind WS-Flows und insbesondere BPEL-Prozesse flexibel im Hinblick auf Änderungen der konkreten Web-Service Endpunkte, die genutzt werden. Jedoch müssen die zu verwendenden Endpunkte i. d. R. beim Deployment eines Prozesses spezifiziert werden. Häufig wäre aber ein spätes Binden an einen Endpunkt anhand bestimmter Kriterien, evtl. sogar instanzbasiert, wünschenswert.

Um hier mehr Flexibilität zu bieten, schlagen [Karastoyanova u. a. (2005a)] eine Erweiterung von BPEL vor, die es erlaubt zur Laufzeit aus bestimmten, im Prozessmodell hinterlegten, Parametern, den zu verwendenden Endpunkt zu bestimmen, evtl. unter Nutzung eines Discovery-Dienstes wie UDDI (vgl. [Bellwood u. a. (2004)]). Prozesse, die diese Erweiterung nutzen, werden auch als *parametrisierte Prozesse* bezeichnet.

Trotz der Möglichkeit, den zu nutzenden Endpunkt zur Laufzeit zu bestimmen, sind parametrisierte Prozesse noch nicht ausreichend flexibel. Es besteht weiterhin eine Bindung an den im Prozessmodell deklarierten Porttyp (und die Operation, die aufgerufen werden soll), die durch die oben beschriebenen Mechanismen nicht beseitigt werden kann. Um Flexibilität hinsichtlich des Porttyps und der Operation zu erreichen, erweitern [Karastoyanova u. a. (2005b, 2006)] das Konzept um die Möglichkeit auch den Porttyp und die Operation zu parametrisieren und zur Laufzeit zu bestimmen. Dies hat den Hintergrund, dass es nach Ansicht der Autoren sehr unwahrscheinlich ist, dass verschiedene Anbieter sich auf denselben Porttyp einigen können, auch wenn es innerhalb einer Domäne definierte Datenformate, wie z. B. RosettaNet etc. existieren. Der Grund für die Unterschiede in den Porttypen kann z. B. die unterschiedlichen Benennungen der Parameter von Operationen oder Unterschiede

im Umfang der bereitgestellten Funktionen sein. Diese Erweiterung bietet *Flexibility by Adaptation*, was die Prozessmodelle vereinfacht und wiederverwendbar macht, erfordert jedoch gemeinsame Datenformate und für die Automatisierung der Adaption semantische Beschreibungen der in Frage kommenden Web-Services.

Die oben beschriebenen Flexibilitätsmechanismen ermöglichen es, zur Laufzeit und instanzbasiert die zu verwendenden Web-Services zu ändern. Die Evolution der Kontrollflussdimension wird jedoch durch diese Ansätze nicht behandelt. Es existieren einige Ansätze zur Lösung dieser Frage, die die Konzepte zur ad-hoc Änderung von Workflows und auch der aspektorientierten Programmierung, die auch schon für allgemeine Workflows genutzt wurden (siehe Abschnitt 2.1.4.1), auf BPEL-Prozesse übertragen.

[Courbis und Finkelstein (2004)] beschreiben einen Ansatz, der auf einer speziellen Implementierung einer BPEL-Engine, die einen dynamischen Aspect-Weaver enthält, aufbaut. Dabei setzen sie auf einen Ansatz, der zwei Arten von Aspekten umfasst. Die sog. *Engine-Aspekte* und die sog. *Prozess-Aspekte*. Engine-Aspekte erweitern die Implementierung der Prozess-Engine um neue Funktionen (beispielsweise Tracing, Debugging-Funktionen oder neue Sprachkonstrukte), indem in Java verfasste Advices dynamisch zur Laufzeit eingewoben werden. Die Prozess-Aspekte modifizieren oder erweitern das Prozessmodell oder eine Prozessinstanz, indem in BPEL verfasste Advices eingewoben werden. Hierdurch lassen sich Aktivitäten ersetzen, der Kontrollfluss modifizieren oder auch Partnerbindungen ändern. Die Wahl von dynamischem Weaving begründen Courbis und Finkelstein damit, dass Hot-Fixes von Prozessen inhärent dynamisch sind und nur zur Laufzeit erfolgen können, da das Abbrechen und Neustarten von Instanzen inakzeptabel ist. Das eigentliche Weaving wird durch Transformation der BPEL-Dokumente realisiert (vgl. [Courbis und Finkelstein (2005a)]). Die Autoren stellen zudem fest, dass diese Transformationen nur zu ganz bestimmten Punkten im Lebenszyklus einer Aktivität vorgenommen werden dürfen, um die Konsistenz zu wahren. Die Praktikabilität des Ansatzes wird anhand einiger Beispiele gezeigt (vgl. [Courbis und Finkelstein (2005b)]).

Einen ähnlichen Ansatz verfolgen auch [Charfi und Mezini (2004)] mit AO4BPEL. Sie beschreiben die Defizite von BPEL, die durch aspektorientierte Programmierung verbessert werden sollen. Zum einen bietet BPEL keine Möglichkeit für dynamische Adaption der Prozesse, was in der hochdynamischen Web-Service Welt mit vielen wechselnden Partner, Business-Rules etc. nachteilig ist. Des Weiteren bietet BPEL laut den Autoren nicht genügend Flexibilität was evolutionäre und ad-hoc Änderungen angeht, da wie oben schon genannt der Ansatz einen Prozess für die Änderung anzuhalten und neu zu starten, für langlaufende Prozesse nicht tauglich ist. Als dritten Punkt nennen die Autoren, dass BPEL keine Unterstützung zur Modularisierung von Crosscutting-Concerns bietet, was dazu führt, dass die Umsetzung der Crosscutting-Concerns über das gesamte Prozessmodell verstreut ist, was die Wartung und die Wiederverwendung der Prozessmodelle verkompliziert. Als Beispiel für solche Crosscutting-Concerns im WS-Flow Umfeld nennen die Autoren die

Nutzung von Business-Rules.

Wie der Ansatz von [Courbis und Finkelstein (2004)] nutzt auch AO4BPEL dynamisches Weaving. Im Unterschied zum oben genannten Ansatz wird hier jedoch nur eine Aspektsprache (konkret BPEL) verwendet. Für das Einweben von Funktionen in die BPEL-Engine schlagen die Autoren sog. *Java code execution Web Services* vor, über die beliebiger Java-Code aufgerufen werden kann. Die Autoren geben keine Regeln für die Korrektheit der Einwebung an, nennen dies jedoch als eine in der Zukunft zu bearbeitende Aufgabe. Auch für diesen Ansatz haben die Autoren die Anwendbarkeit untersucht (vgl. [Charfi und Mezini (2005a, b)]).

Ein weiterer Ansatz auf diesem Gebiet findet sich bei [Erradi u. a. (2005)] mit dem AdaptiveBPEL Framework. Ziel des AdaptiveBPEL-Systems ist es ein Framework zur dynamischen Adaption von Web-Service Kompositionen zu schaffen. Dabei soll ein Weg geschaffen werden, der dynamische und instanzbasierte Adaption für Web-Service Orchestrierungen erlaubt, insbesondere um flexibel auf sich ändernde Business-Rules zu reagieren, oder die *Quality of Services* (QoS) an sich ändernde Policies (siehe Abschnitt 2.4) anzupassen. Der Ansatz nutzt dynamisches Weben um zur Laufzeit instanzbasiert anhand von Policies selektierte Aspekte einzuweben.

[Baresi und Guinea (2005)] beschreiben eine Einsatzmöglichkeit von AOP für BPEL-Prozesse, die Nutzung zu Monitoring Zwecken. Die Autoren stellen fest, dass in konventionellen Ansätzen Geschäfts- und Monitoring-Aktivitäten in einem Prozess vermischt sind, also keine Separation-of-concerns vorliegt. Der Ansatz ist nun, Monitoring-Aktivitäten vor und nach den Geschäftsaktivitäten einzuweben. Hierzu schlagen die Autoren statisches Weaving durch einen Präprozessor vor. Hierdurch wird eine gute Modularisierung der eigentlichen Logik des Geschäftsprozesses und der Monitoring-Logik erreicht.

Ein weiterer, AOP-ähnlicher Ansatz findet sich bei [Rosenberg und Dustdar (2005)]. Hier kommen AOP-Techniken zum Einsatz um Business-Rules in einen Geschäftsprozess einzuweben. Hintergrund ist die Beobachtung, dass sich Business-Rules wesentlich schneller ändern als Geschäftsprozesse. Der Ansatz sieht nun vor, dass vor und nach Aktivitäten Aufrufe eines als Web-Service ausgeführten Business-Rule Brokers eingewoben werden, der die hinterlegten Business-Rules ausführt. Dieser Ansatz wird von den Autoren zwar nicht mit AOP in Beziehung gesetzt, ist jedoch sehr eng mit den oben genannten AOP-Ansätzen verwandt.

2.2 Aspekt-Orientierte Programmierung

2.2.1 Modularisierung von Softwaresystemen

Ein Ziel beim Entwurf von Softwaresystemen ist es, eine möglichst hohe Modularität zu erreichen, da modulare Systeme hinsichtlich Erstellung, Wartung, Verständnis etc. vorteilhaft sind. Klassische Ansätze zur Modularisierung sind Konstrukte wie Module (in prozeduralen Programmiersprachen) oder Klassen (in objektorientierten Sprachen), die von der genutzten Programmiersprache selbst bereitgestellt werden. Nach [Kiczales u. a. (1997)] kann man die Grundlage all dieser Mechanismen als eine Art „generalisierter Prozedur“ ansehen. Eine Folge dieser gemeinsamen Wurzel ist, dass die zur Entwicklung von Softwaresystemen in solchen Sprache eingesetzten Entwurfs-Methoden überwiegend eine funktionale Zerlegung (vgl. [Parnas (1972)]) ergeben.

Objektorientierte Ansätze haben durch die Kapselung von Daten und Verhalten in Klassen eine Verbesserung der Modularität gegenüber prozeduralen Ansätzen erreicht (vgl. [Murphy und Schwanninger (2006)]). Häufig kommt es dabei jedoch vor, dass zwei konkurrierende Anforderungen an das System verschiedene Zerlegungen für den Entwurf nahelegen. Eine solche Anforderung, die sich nicht sinnvoll durch funktionale Zerlegung modularisieren lässt und trotzdem mit der funktional zerlegten Kernfunktionalität koordiniert werden muss, nennt man *Crosscutting-Concern* („querschnittliche Funktion“). Da gängige prozedurale und OO-Sprachen nur einen einzigen Kompositionsmechanismus (i. d. R. funktionale Zerlegung) unterstützen (vgl. [Kiczales u. a. (1997)]), können solche Crosscutting-Concerns nicht sinnvoll modularisiert werden, so dass die Co-Komposition vom Entwickler selbst vorgenommen werden muss. Das bedeutet, dass der Code zur Realisierung solcher Anforderungen potentiell über den Quellcode der gesamten anderen Funktionen verstreut werden muss, was offensichtlich im Hinblick auf Wartbarkeit, Verständlichkeit etc. sehr nachteilig ist. Einfache Beispiele für solche Anforderungen sind transaktionales Verhalten oder Logging.

2.2.2 Grundidee der aspektorientierten Programmierung

Ein übliches Vorgehen beim objektorientierten Entwurf ist es, gemeinsame Funktionalitäten in verschiedenen Klassen zu finden und diese Funktionalitäten in der Vererbungshierarchie zur Wurzel hin zu verschieben. Dieses Vorgehen ist für Crosscutting-Concerns nicht anwendbar (vgl. [Elrad u. a. (2001)]).

Um dieses Problem zu lösen, wurde ein neues Programmierparadigma, die aspektorientierte Programmierung vorgeschlagen, das es erlaubt, Crosscutting-Concerns orthogonal aus

der Vererbungshierarchie herauszuziehen und in einem neuen Konstrukt zur Modularisierung, den sog. *Aspekten* zu realisieren. Aspekte bilden in aspektorientierten Sprachen einen weiteren erstrangigen Dekompositionsmechanismus, der parallel zur Dekomposition in Prozeduren, Methoden etc. existiert. [Kiczales u. a. (1997)] schlagen für diese klassischen Einheiten der Zerlegung den Begriff *Komponente* vor. Für das Design wird nun vorgeschlagen, dass Funktionalitäten, die durch funktionale Zerlegung abgebildet werden können, als Komponenten und alle anderen Funktionalitäten als Aspekte realisiert werden. Damit ist bei aspektorientierter Programmierung die Frage, nach welchem Kriterium ein System am besten zerlegt werden soll, nicht mehr entscheidend, da man mehrere Kriterien zugleich nutzen kann (vgl. [Cilia u. a. (2003)]).

Eine AOP-Implementierung besteht aus einem Komponentenprogramm und einem oder mehreren Aspektprogrammen. Hierbei wird für die Komponentenprogramme und die Aspektprogramme nicht zwingend dieselbe Sprache verwendet. Aus diesen Bestandteilen erzeugt der sog. *Aspect Weaver* das endgültige Programm, indem er die Aspekte in die Komponenten „einwebt“.

2.2.3 Bestandteile einer aspektorientierten Programmiersprache

[Chavez und Lucena (2003)] beschreiben ein Meta-Modell für aspektorientierte Programmiersprachen. Nach diesem Modell ist eine aspektorientierte Programmiersprache durch folgende Punkte gekennzeichnet:

- Komponenten-Modell
- Joinpoint-Modell
- Core-Modell
- Weaving-Modell
- Quantifizierungsmechanismus
- Obliviousness-Eigenschaft

Hierbei beschreibt das Komponenten-Modell, welche Sprache für die Beschreibung der Komponentenprogramme verwendet wird und welche primäre Zerlegungsart zur Modularisierung der Komponenten verwendet wird. Damit wird auch festgelegt, was eine Komponente ist, welche Kompositionsprinzipien für Komponenten existieren und welchen Regeln deren Komposition folgt. Diese Eigenschaft ist der Grund, weshalb AOP andere Programmierparadigmen nicht ersetzt, sondern ergänzt. Ein Beispiel für ein solches Komponenten-Modell ist das Objekt-Modell, das die Grundlage der OOP bildet.

Das Joinpoint-Modell beschreibt, an welchen Stellen, *Joinpoints* genannt, innerhalb einer Komponente Aspekte eingewebt werden können, welche Arten von Joinpoints existieren und definiert welche Beschränkungen für ihre Benutzung existieren. Das Joinpoint-Modell ist in seinem Gestaltungsspielraum durch das Komponenten-Modell beschränkt. Einige beispielhafte Joinpoints für das oben beschriebene Komponenten-Modell auf Basis des Objekt-Modells wären Methodenaufrufe, Feldzugriffe etc.

Das Core-Modell bildet ein Framework zur Beschreibung von Aspekten als Abstraktionsmechanismus und eines zugehörigen Kompositionsmechanismus. Ein Aspekt spezifiziert hierbei eine Erweiterung (dieser Begriff umfasst hier sowohl die Ergänzung, als auch das Entfernen von Funktionalität) und eine Menge von Joinpoints, an denen die Erweiterung eingewoben werden soll. Der Mechanismus um Aspekte mit Komponenten zu komponieren wird in diesem Kontext als *Crosscutting* bezeichnet.

Das Weaving-Modell beschreibt die Art und Weise, wie das Einweben der Aspekte in die Komponenten stattfindet. Insbesondere beschreibt es, ob der ursprüngliche Programmcode nach dem Weben noch vorhanden ist, und zu welchem Zeitpunkt (Übersetzungszeit, Laufzeit etc.) das Weben stattfindet.

Der Quantifizierungsmechanismus beschreibt eine Methode, um eine Menge von Joinpoints durch einen Ausdruck zu spezifizieren. Das bedeutet, dass es möglich ist, für einen Aspekt eine beliebige Menge von Komponenten zu spezifizieren, in die der Aspekt eingewoben werden soll. Der Quantifikationsmechanismus muss hierzu eine Sprache bereitstellen, in der die Ausdrücke gebildet werden. Dies ist erforderlich, um eine flexible Komposition von Aspekten und Komponenten zu erlauben.

Die Obliviousness-Eigenschaft beschreibt, dass eine Komponente keinerlei Wissen über die eingewobenen/einzuwebenden Aspekte verfügt. Das bedeutet, dass bei der Entwicklung der Komponente keine Vorkehrungen zur Unterstützung von Aspekten getroffen werden müssen. Dies ist bedeutsam, da hierdurch einerseits die Anwendung von AOP auf bereits existierende Systeme ermöglicht und andererseits die separate Testbarkeit der Kernfunktionalität in den Komponenten erreicht wird.

2.2.4 Weaving-Varianten

Das Einweben der Aspektprogramme in die Komponentenprogramme kann generell betrachtet auf zwei Arten geschehen: statisch oder dynamisch. *Statisches Weaving* bedeutet in diesem Zusammenhang, dass die Aspekte zur Laufzeit fest mit dem Komponentenprogramm verbunden und nicht zur Laufzeit austauschbar oder beeinflussbar sind. Im Gegensatz dazu erlaubt es *dynamisches Weaving* Aspekte zur Laufzeit zu (de-)aktivieren, auszutauschen etc. In Hinblick auf diese Eigenschaft spricht man auch von *statischer* AOP

und *dynamischer* AOP.

Der klassische Ansatz beim statischen Weaving ist es, den Quellcode, Bytecode oder Binärcode (im Rest dieses Abschnitts vereinfachend als Code bezeichnet) des Komponentenprogramms zu instrumentieren (vgl. [Cilia u. a. (2003)]). Es wird an den identifizierten Joinpoints der passende Aspektcode (und falls nötig zusätzlicher Code zum überprüfen der dynamischen Bedingungen bei dynamischen Joinpoints) direkt eingesetzt oder Aufrufe (sog. Hooks) des Aspektcodes eingesetzt.

Für das dynamische Weaving gibt es mehrere Ansätze (vgl. [Sato u. a. (2003)]). Zum einen kann auch hier der Code mit Hooks instrumentiert werden. Im Gegensatz zum statischen Weben, bei dem die relevanten Joinpoints bekannt sind, muss hierbei jeder potentielle Joinpoint mit einem Hook versehen werden. Die Hooks signalisieren dann während der Laufzeit das Erreichen eines Joinpoints einer speziellen Komponente (bei [Cilia u. a. (2003)] Reaction Manager genannt), die dann wiederum die passenden Aspekte ausführt. Zum anderen gibt es die Möglichkeit ohne Veränderungen des Codes auszukommen und das erreichen der Joinpoints zur Laufzeit durch einen sog. Interceptor zu erkennen. Hierfür ist die Unterstützung der Ausführungsumgebung notwendig. Zum einen kann hierzu eine modifizierte Virtuelle Maschine oder eine Debugger-Schnittstelle oder bei interpretierten Sprachen ein modifizierter Interpreter (siehe [Popovici u. a. (2002); Sato u. a. (2003)]) verwendet werden. Zum anderen können die Ausführungsumgebungen auch von Anfang an so gebaut werden, dass sie die passenden Ereignisse beim erreichen eines potenziellen Joinpoints nach außen publizieren (vgl. [Cilia u. a. (2003)]). Abbildung 2.3 veranschaulicht die verschiedenen Konzepte nochmals.

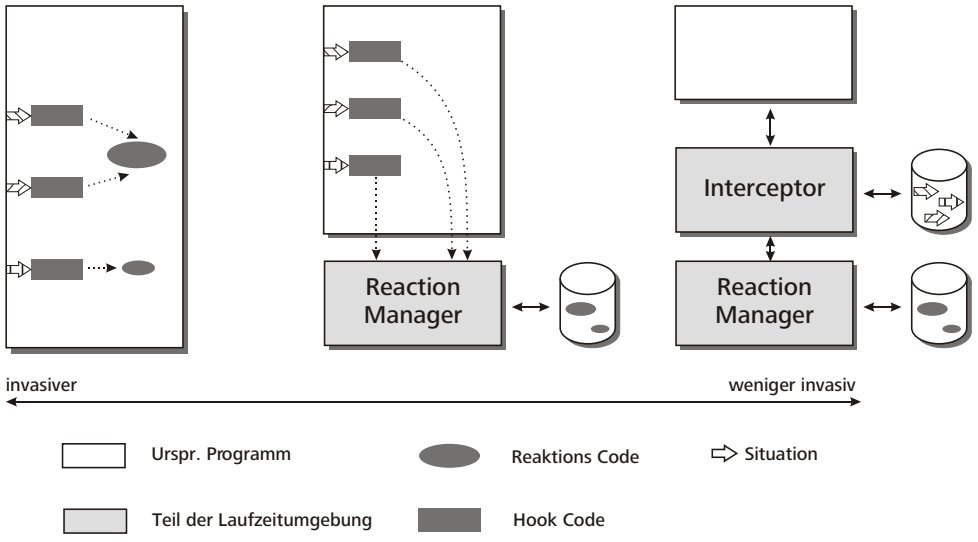


Abbildung 2.3: Invasivität verschiedener Weaving-Mechanismen (nach [Cilia u. a. (2003)])

Der Einsatz von dynamischem Weaving bietet einige Vorteile gegenüber statischem Wea-

ving, was jedoch i. d. R. durch eine schlechtere Performance erkauft wird. Die Vorteile des dynamischen Weaving sind u. a. kurze Turnaround-Zeiten, was schnellere Design-Testen-Zyklen für Aspekte ermöglicht. Dies ist u. a. für das Prototyping von Vorteil, wo nachdem sich die Aspekte stabilisiert haben, auf ein statisches Weaving Verfahren gewechselt werden kann, um die Performance-Nachteile zu vermeiden (vgl. [Popovici u. a. (2002)]).

Es ist anzumerken, dass das verwendete Weaving-Modell keinen Einfluss auf die Art der Joinpoints (statisch oder dynamisch), die eine AOP-Implementierung unterstützt, hat. Auch in einer statischen AOP-Sprache können dynamische Joinpoints unterstützt werden, da es möglich ist statisch Code für die Prüfung der dynamischen Bedingungen an allen potentiell in Frage kommenden Joinpoints einzuweben. Die Dimensionen Weaving und Joinpoints sind also orthogonal, da sie unabhängig voneinander ausgeprägt werden können.

2.2.5 Das Programmiermodell von AspectJ

AspectJ (vgl. [Kiczales u. a. (2001)], [Gradecki und Lesiecki (2003)]) ist eine der meistgenutzten aspektorientierten Programmiersprachen und diente als Vorlage für eine Reihe anderer aspektorientierter Programmiersprachen, weshalb es hier vorgestellt wird. Es verwendet Java als Komponentensprache, ist also in der Lage bestehende Java-Programme durch Aspekte zu erweitern. Die Aspekte und Erweiterungen werden in einer Sprache beschrieben, die auf Java basiert und dessen Sprachumfang um Konstrukte zum Beschreiben von Aspekten, zur Quantifizierung etc. erweitert.

AspectJ enthält fünf Kernkonzepte:

- Aspekte
- Joinpoints
- Pointcuts
- Advices
- Inter-Type Deklarationen

Aspekte sind in AspectJ Module um mehrere zusammengehörige Pointcuts, Advices und Inter-Type Deklarationen zu kapseln. Sie weisen viele Ähnlichkeiten zu Java-Klassen auf, was die Definition und die Verwendung angeht, so gibt ist es z. B. einen Vererbungsmechanismus für Aspekte.

Joinpoints sind in AspectJ hauptsächlich statisch definiert und umfassen z. B. Methodenaufruf, -ausführung, Zugriffe auf Attribute, Aufruf von Konstruktoren etc. Zusätzlich gibt es noch dynamische Joinpoints, die Prädikate über den Kontrollfluss anbieten (z. B. Methode A wird aus Methode B aufgerufen).

Pointcuts spezifizieren eine Menge von konkreten Joinpoints an denen ein Aspekt eingewoben werden soll. Durch Nutzung von regulären Ausdrücken erlauben Pointcut-Ausdrücke auch die Quantifizierung von Joinpoints. Ein Beispiel für eine solche Quantifizierung ist ein Pointcut-Ausdruck, der den Aufruf aller Methoden, die auf die Signatur `public String get*()` matchen, also alle Methoden, deren Name mit `get` beginnt, ein `String` Objekt zurückgeben und keine Parameter haben, spezifiziert.

Advices spezifizieren die Erweiterungen, die an den durch die Pointcuts spezifizierten Stellen eingewoben werden sollen. Es gibt verschiedene Arten von Advices, die festlegen, wann die Erweiterungen ausgeführt werden. So wird es möglich zu spezifizieren, dass eine Erweiterung vor (Before-Advice) oder nach (After-Advice) dem Ausführen einer Methode ausgeführt wird. Ebenso ist es möglich, eine Aktion die Methode umschließen zu lassen (Around-Advice), d. h. die Erweiterung führt einen Teil ihres Codes aus, bis eine spezielle Anweisung folgt, die den Aufruf der umschlossenen Methode auslöst und nach der Rückkehr aus der Methode den Rest des Codes der Erweiterung ausführt. Auf dieselbe Art und Weise lassen sich auch komplette Implementierungen ersetzen, indem man einen Around-Advice definiert, der keinen Aufruf der umschlossenen Methode enthält.

Zusätzlich zu den zuvor genannten Bestandteilen, die für sich alleine genommen schon die in Abschnitt 2.2.3 beschriebenen Kriterien erfüllen, bietet AspectJ auch sog. *Inter-Type Deklarationen* an. Diese sind eine Möglichkeit, bestehende Java-Klassen um Methoden- und Feld-Definition zu erweitern. Dies ermöglicht Crosscutting-Concerns, die Modulstrukturen betreffen, zu modularisieren. Dieses Konzept ist auch als Open-Classes bekannt.

Für das Weaving der Aspekte bietet AspectJ drei Optionen:

- Nutzung eines speziellen erweiterten Java-Compilers
- Bytecode-Modifikation von bereits kompilierten Java-Klassen in JAR-Archiven
- Bytecode-Modifikation beim Laden einer Klasse in die JVM durch einen speziellen Classloader.

In allen Fällen findet statisches Weaving statt, da die Aspekte nicht zur Laufzeit ausgetauscht oder entfernt werden können. Auch die dynamischen Joinpoints werden durch das statische Weaving abgedeckt, indem Code erzeugt wird, der zur Laufzeit die dynamische Bedingung prüft.

2.3 WS-Notification

2.3.1 Grundlagen von Publish/Subscribe Systemen

Publish/Subscribe (oder kurz *Pub/Sub*) ist ein Kommunikationsparadigma in verteilten Systemen. Publish/Subscribe ist verwandt mit dem *Message-Queueing* Paradigma und ist i. d. R. ebenfalls Teil einer *Message-Orientied-Middleware* (MOM).

Generell besteht ein Pub/Sub-System aus einem oder mehreren *Publishern*, die Nachrichten oder Ereignisse publizieren, einem oder mehreren *Subscribern*, die Nachrichten zu einem bestimmten Sachverhalt abonnieren und einem zwischengeschalteten *Broker*, der für die Verteilung der publizierten Nachrichten an die interessierten Subscriber sowie für die Verwaltung der „Abonnements“ (*Subscriptions*) zuständig ist. Publish/Subscribe Systeme realisieren also eine n:m-Kommunikation, ähnlich Multicasts.

Abhängig von der zugrundeliegenden MOM bieten Publish/Subscribe Systeme eine Reihe von Qualities of Services. Die Häufigsten sind Persistenz (Nachrichten gehen auch bei Systemausfällen nicht verloren), Prioritäten (Nachrichten können mit verschiedenen Prioritäten versendet und empfangen werden), Transaktionalität (Nachrichten können innerhalb einer Transaktion verarbeitet werden) und Zuverlässigkeit (Reliability, Nachrichten werden trotz Fehlern an alle Empfänger zugestellt).

2.3.1.1 Subscriptions-Schemata

In der Praxis sind i. d. R. nicht alle Subscriber an allen Nachrichten interessiert, sondern nur an den für sie relevanten. Um die Subscriber von der Aufgabe der Nachrichtenfilterung zu entlasten und die Skalierbarkeit zu erhöhen, ist es sinnvoll einen Mechanismus zum Ausdrücken des Interesses an bestimmten Nachrichten für Subscriber einzuführen und die Filterung im Broker vorzunehmen. Es gibt mehrere verbreitete Varianten, wie die Verteilung der Nachrichten an die Subscriber strukturiert sein kann (vgl. [Eugster u. a. (2003)]):

- Topicbasiert
- Inhaltsbasiert (auch content-based genannt)
- Typbasiert

Beim topicbasierten Ansatz ordnet der Publisher jede Nachricht einem *Topic* (Thema) zu. Die Subscriber können sich für einen oder mehrere Topics anmelden und bekommen vom Broker alle Nachrichten, die auf diesem Topic publiziert wurden, zugestellt. Hierbei werden die Topics über eindeutige Namen identifiziert. Teilweise werden auch hierarchisch

strukturierte Namensräume für die Topics angeboten, wobei bei Subskription auf einen Topic auch automatisch alle Nachrichten auf seinen Kind-Topics zugestellt werden. Topic-basierte Systeme sind statisch und bieten wenig ausdrucksstarke Möglichkeiten bei der Spezifikation von Subskriptionen, jedoch sind sie sehr effizient und performant.

Beim inhaltsbasierten (*content-based*) Ansatz werden keine vordefinierten externen Kriterien wie z. B. Topic-Namen verwendet, sondern auf den Inhalt der Nachricht selbst zurückgegriffen. Hierbei können entweder die in der Nachricht enthaltenen Daten selbst oder die mit der Nachricht verknüpften Meta-Daten verwendet werden. Die Subscriber beschreiben in diesem Fall ihr Interesse an bestimmten Nachrichten durch Boolesche Ausdrücke und logische Operatoren, die Nachrichteninhalte referenzieren. Dieses Schema besitzt eine sehr hohe Ausdruckskraft und erlaubt sehr feingranulare Subskriptionen, verursacht jedoch einen erheblichen Performance-Overhead zur Laufzeit. Es gibt auch hybride Verfahren, die inhaltsbasierte Subskriptionen auf Topics anbieten, wobei sog. Selektoren verwendet werden, um die Nachrichteninhalte zu spezifizieren.

Beim typbasierten Ansatz werden verschiedene Typen von Nachrichten definiert. Dieser Ansatz stammt daher, dass Topics oft nicht nur nach Inhalt sondern auch nach Nachrichtenstrukturen gliedern, so dass es nahe liegt direkt die Struktur der Nachricht zu verwenden. Hierdurch wird es auch möglich, bereits zur Kompilierzeit Typ-Sicherheit zu bieten.

2.3.1.2 Entkopplung durch Publish/Subscribe

Das Publish/Subscribe Paradigma bietet einen hohen Grad an Entkopplung zwischen den Kommunikationspartnern (vgl. [Eugster u. a. (2003)]). Es bietet folgende drei Arten der Entkopplung:

- Zeitliche Entkopplung
- Räumliche Entkopplung
- Synchronisations-Entkopplung

Zeitliche Entkopplung bedeutet in diesem Zusammenhang, dass die Kommunikationsparteien nicht zeitgleich an der Interaktion teilnehmen müssen. Es ist also möglich, dass zum Zeitpunkt des Veröffentlichens einer Nachricht keiner der Subscriber online ist und ebenso zum Zeitpunkt der Zustellung der Nachricht an einen bestimmten Subscriber der Publisher nicht online ist.

Räumliche Entkopplung bedeutet, dass die an der Kommunikation beteiligten Parteien keinerlei Wissen über die andere Partei haben müssen. Es ist also möglich, dass ein Publisher nicht weiß, welche Subscriber die von ihm veröffentlichten Nachrichten empfangen werden.

Ebenso ist es nicht zwingend notwendig, dass die Subscriber die Identität des Publishers einer Nachricht kennen. Ebenso ist es nicht zwingend erforderlich, dass der Broker die Identität aller Publisher und Subscriber kennen muss, da es möglich ist Netze von Brokern zu erzeugen, in denen die einzelnen Broker wechselseitig als Publisher und Subscriber auftreten.

Synchronisations-Entkopplung bedeutet, dass die Kommunikation asynchron verlaufen kann, also keine Blockierung beim Senden und Empfangen einer Nachricht eintritt. Das bedeutet, dass die Verarbeitung der Nachrichten nicht im Hauptkontrollfluss der Kommunikationspartner stattfindet.

2.3.2 Die WS-Notification Protokollfamilie

Für eine Reihe von Anwendungen ist ein ereignisbasiertes Interaktionsschema eine wesentlich natürlichere Art der Kommunikation als Request/Response. Der fundamentale Unterschied zwischen Request/Response und ereignisbasierten Interaktionsformen ist, dass bei ereignisbasierter Kommunikation das Ereignis selbst in den Mittelpunkt rückt und die Aufrufe zu seiner Signalisierung verdeckt werden.

Mit dem Ziel ereignisbasierte Kommunikation auch in der Web-Service Welt zur Verfügung zu stellen, wurde von der OASIS die Familie der *Web Services Notification* (WS-Notification oder kurz WS-N) Spezifikationen definiert (vgl. [Graham u. a. (2004a)]). Der Standard definiert einen interoperablen Weg um Ereignisse (Events) und Benachrichtigungen (Notifications) zwischen Web-Services auszutauschen. WS-Notification besteht auf drei Teilspezifikation, die aufeinander und auf etablierten Web-Service Standards wie SOAP, WSDL, WS-Addressing und dem WS-ResourceFramework aufbauen:

- WS-BaseNotification
- WS-Topics
- WS-BrokeredNotification

WS-Notification definiert eine eigene Terminologie, die teilweise von anderen im Pub/Sub Umfeld abweicht und für das Verständnis des Folgenden wichtig ist.

Situation Das Auftreten eines Vorkommnisses (hier wird bewusst nicht von einem „Ereignis“ gesprochen) innerhalb eines Web-Service oder seiner Umgebung, das von Interesse für Dritte ist.

Notification Die Übertragung einer ein-weg Nachricht (in diesem Zusammenhang als „Notification Message“bezeichnet) mit Informationen über eine Situation.

Notification Producer Rollenbezeichnung für einen Web-Service, der Instanzen von Notification Messages erzeugt und verteilt. Ein Notification Producer kann entweder ein Publisher oder ein Notification Broker sein.

Notification Consumer Ein Web-Service, der Notification Nachrichten von einem Notification Producer empfängt.

Subscription Eine Subscription ist eine WS-Resource (siehe WS-ResourceFramework), die die Beziehung zwischen einem Notification Producer, einem Notification Consumer und einem Topic, sowie einigen Meta-Daten darstellt.

Subscriber Eine Entität (häufig ein Web-Service), der eine Subscription bei einem Notification Producer erzeugt. Ein Subscriber ist nicht zwingend der Notification Consumer, für den die Subscription eingerichtet wird.

Publisher Ein Publisher ist eine Entität (nicht zwingend ein Web-Service), die Situationen beobachtet und in Notification Messages umsetzen kann. Ein Publisher muss nicht zwingend ein Notification Producer sein.

Subscription Manager Ein Web-Service, der Funktionen zum Abfragen und Manipulieren von Subscription-Resourcen bereitstellt.

Notification Broker Ein Notification Broker ist ein vermittelnder Web-Service, der Notification Consumer von Publishern entkoppelt. Ein Notification Broker agiert sowohl als Notification Producer, als auch als Notification Consumer.

Topic Ein Topic wird zum Kategorisieren von Notifications nach bestimmten Kriterien verwendet (siehe Abschnitt 2.3.1.1).

Topic Tree Eine hierarchische Gliederung von Topics.

Topic Space Ein Wald aus Topic Trees, die alle demselben Namensraum zugeordnet sind.

2.3.2.1 WS-BaseNotification

WS-BaseNotification (vgl. [Graham u. a. (2004b)]) bildet die Grundlage der WS-Notification Protokollfamilie. Es definiert die grundlegenden Rollen und Nachrichten, die zwischen den teilnehmenden Partnern ausgetauscht werden (vgl. [Niblett und Graham (2005)]). Die in WS-BaseNotification beteiligten Rollen sind Publisher, Notification Producer, Notification Consumer, Subscriber und Subscription Manager. Abbildung 2.4 gibt einen Überblick über WS-BaseNotification.

Diesen Rollen werden von WS-BaseNotification definierte Schnittstellen (Porttypen) zugewiesen, die Web-Services, die diese Rollen einnehmen, implementieren müssen. So müssen alle Notification Producer einen bestimmten Porttyp implementieren, der Funktionen zum Abfragen der unterstützten Topics und Topic Dialekte (siehe Abschnitt 2.3.2.2)

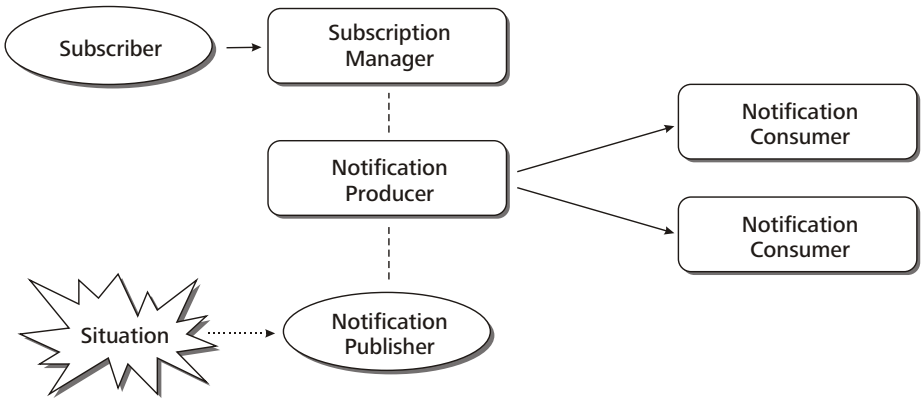


Abbildung 2.4: Überblick über WS-BaseNotification (in Anlehnung an [Niblett und Graham (2005)])

sowie der zuletzt versendeten Notification Nachricht anbietet. Ebenso müssen alle Notification Consumer einen bestimmten Porttyp implementieren, der genutzt wird, um Notification-Nachrichten zuzustellen. Der Subscription Manager wiederum muss einen Porttyp implementieren, der zum Registrieren und Verwalten einer Subscription dient.

Der Ablauf der Interaktion bei WS-BaseNotification ist folgender:

1. Der Subscriber registriert eine Subscription beim SubscriptionManager (i. d. R. ist dies derselbe Web-Service wie der Notification Producer, die Spezifikation erlaubt jedoch auch Ausnahmen). Dabei erhält er eine Endpunkt-Referenz auf die Subscription-Resource zurück, die zur Verwaltung der Subscription genutzt werden kann.
2. Um eine neue Notification zu verteilen sendet der Notification Producer an alle registrierten Consumer eine one-way Nachricht, die die Notification Nachricht enthält.

Bei WS-BaseNotification findet also eine direkte Interaktion zwischen Producer und Consumer statt.

WS-BaseNotification beschreibt ein Nachrichtenformat, das als Envelope für die anwendungsspezifischen Nachrichten genutzt werden kann, erlaubt jedoch auch, dass Nachrichten ohne diesen Envelope versendet werden können. Das zu verwendende Format wird durch den Subscriber festgelegt. Das Format der anwendungsspezifischen Nachrichten (die mögliche Einbettung in den Envelope spielt hierbei keine Rolle) wird in einem WSDL-Dokument als Bestandteil des Porttyps des Consumers definiert.

Zur Filterung der Nachrichten an die Consumer definiert WS-BaseNotification drei Arten von Filter-Ausdrücken (vgl. [Niblett und Graham (2005)]):

Topic-Filter Ein Topic-Filter beschreibt, für welche Topics ein Consumer Notifications empfangen soll. Nähere Details zu Topics und Topic-Filtern finden sich in Abschnitt 2.3.2.2.

Message-Filters Ein Message-Filter ist ein boolescher Ausdruck über dem Inhalt der Notification Nachricht. Er entspricht also dem content-based Filtering Ansatz aus Abschnitt 2.3.1.1.

Producer state Filter Der Filter ist ein Prädikat über den Zustand des Producers, wobei der Zustand nicht in der Notification Nachricht enthalten ist. Das bedeutet, dass der Subscriber Informationen über den Aufbau und die Eigenschaften des Producers besitzen muss.

Nach [Niblett und Graham (2005)] kann WS-BaseNotification als Umsetzung des *Notification Pattern* angesehen werden. Das Notification Pattern ist eine Adaption des Observer-Patterns (vgl. [Gamma u. a. (1995)]) für die SOA, wobei die Ideen des eventbasierten Programmierens mit denen der SOA verbunden werden. Die Charakteristiken des Patterns sind die folgenden:

- Dienste, die Notifications verarbeiten können, registrieren sich bei Diensten, die Notifications erzeugen können.
- Der Notification Producer verteilt Notifications durch one-way Nachrichten an alle registrierten Consumer.
- Der Producer kann eine beliebige Anzahl von Nachrichten an die Consumer verteilen.

Der Unterschied zum klassischen Observer-Pattern liegen darin, dass anstelle von Zustandsänderungen des Subjekts, beliebige Informationen durch den Producer verteilt werden können, dass an die Stelle der expliziten Deregistrierung ein zeitbasiertes Ablaufen der Subscriptions tritt und dass das Notification Pattern Filtering von Nachrichten vorsieht.

2.3.2.2 WS-Topics

WS-BaseNotification macht keine Aussagen zur Syntax und Struktur von Topics. Dieses Themengebiet wird in der WS-Topics Spezifikation (vgl. [Graham u. a. (2004d)]) behandelt, die verschiedene Dialekte für die Selektion von Topics und ein XML-Modell für Meta-Daten zu Topics definiert.

WS-Topics definiert einen Mechanismus, um Topics eindeutige Namen zuzuweisen. Hierzu werden Topics in XML-Namespaces eingruppiert. Dabei können Mengen von Topics denselben Namespace nutzen. Die Namespaces sind nicht an einen bestimmten Notification Producer gebunden und können wiederverwendet werden. WS-Topics bietet die

Möglichkeit Topics hierarchisch in einem Topic-Baum zu gliedern und aus mehreren solchen Topic-Bäumen einen Wald (Topic-Space) zu bilden, wobei alle Topics des Topic-Space im selben Namespace liegen müssen. Die Auswahl von Topics innerhalb der Hierarchie geschieht über ein strukturiertes Namens-Schema, wobei ein Topic über die Kombination aus Namespace und lokalem Namen adressiert wird. Der Name eines Topics muss nicht zwingend innerhalb einer Notification Nachricht erscheinen.

Zur Selektion von Topics definiert WS-Topics drei verschiedene sog. *Topic-Dialekte* unterschiedlicher Mächtigkeit:

Simple Topic Expression Der Topic wird über seinen XML Qualified Name angesprochen. Dieser Dialekt erlaubt nur Wurzel-Topics innerhalb eines Topic-Space anzusprechen.

Concrete Topic Path Expressions Bei diesem Dialekt wird der Topic-Namespace und eine Sequenz von durch Schrägstriche getrennten Topic-Namen verwendet

Full Topic Path Expression Dieser Dialekt verwendet eine eingeschränkte Variante von XPath um Topics innerhalb eines Topic-Space zu selektieren. Bei diesem Dialekt sind Wildcard-Zeichen erlaubt (`*`, `.` und `//`) sowie konjunktive Verknüpfungen von Ausdrücken, wodurch es ermöglicht wird, mehr als einen Topic direkt über eine Topic-Expression auszuwählen. Full Topic Path Expressions sind eine Generalisierung der Concrete Topic Path Expressions.

Darüber hinaus sind noch weitere Topic-Dialekte erlaubt. Die Entscheidung, welche Dialekte unterstützt werden obliegt dem Notification Producer.

Zur näheren Beschreibung von Topics und Topic-Spaces definiert WS-Topics ein XML-basiertes Meta-Modell, das es erlaubt zu spezifizieren, welche Nachrichtentypen auf einem Topic publiziert werden dürfen, die hierarchischen Beziehungen zwischen Topics festzulegen und das auch einen Aliasing-Mechanismus für Topics bietet. Das Meta-Modell sieht explizit Erweiterungsmechanismen vor, so dass das Modell für spezifische Anforderungen erweitert werden kann.

2.3.2.3 WS-BrokeredNotification

Bei WS-BaseNotification findet de facto keine Unterscheidung zwischen Notification Producer und Publisher statt, was bedeutet, dass der Notification Producer ebenfalls für die Verwaltung von Topics und Subscriptions, sowie die Übertragung der Notifications an die entsprechenden Consumer zuständig ist. WS-BrokeredNotification (vgl. [Graham u. a. (2004c)]) baut auf den von WS-BaseNotification definierten Rollen und Nachrichten auf und führt den Notification Broker als Mittler zwischen Producern und Consumern ein. Hierbei wird die Identifikation der für eine Notification relevanten Subscriptions vom

Producer auf den Broker verlagert. Der Notification Broker ist ebenfalls ein Web-Service, der sowohl den Notification Publisher als auch den Notification Consumer Porttyp implementiert. Hierdurch kann er auf einem anderen Knoten als Publisher und Consumer betrieben werden. Ebenso übernimmt er i. d. R. die Aufgabe des Subscription Managers. Abbildung 2.5 gibt einen Überblick über WS-BrokeredNotification.

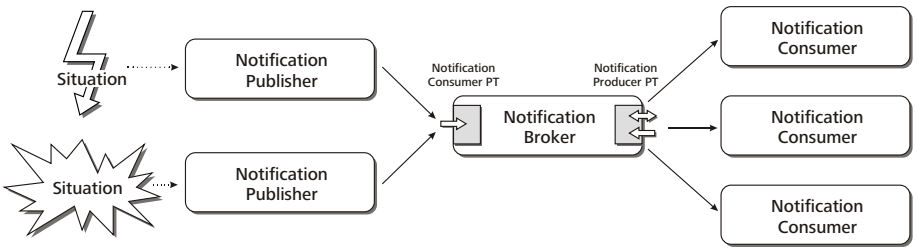


Abbildung 2.5: Überblick über WS-BrokeredNotification (die Abkürzung „PT" steht für Porttyp)

Da es teilweise für den Publisher mit großem Aufwand verbunden ist, eine Situation zu beobachten oder eine Notification Nachricht dafür aufzubauen (man denke an Embedded-Systeme), bietet WS-BrokeredNotification die Möglichkeit, dass ein Broker optional das sog. *demand-based publishing* (vgl. [Niblett und Graham (2005)]) unterstützt. Hierbei fungiert der Notification Publisher auch als Notification Producer, jedoch mit genau einem Consumer, dem Notification Broker. Dieser registriert sich beim Publisher, wenn ein Consumer sich bei ihm für einen Topic registriert. So kann ein Publisher auch in einer Umgebung mit Broker feststellen, ob es erforderlich ist, dass er eine Notification Nachricht erzeugt, oder nicht.

WS-BrokeredNotification bietet gegenüber WS-BaseNotification einige Vorteile (vgl. [Niblett und Graham (2005)]):

- Reduktion der Zahl der Verbindungen und Referenzen zwischen Services.
- Broker fungiert als eine Art „Finder-Service" der Consumer und Publisher in Verbindung bringt (Auffinden über Topic, Nachrichteninhalte etc.).
- „Anonyme" Notification, d. h. Publisher und Consumer müssen einander nicht kennen.
- Broker kann zusätzliche Funktionalitäten bieten (z. B. Logging, Nachrichten- oder Topic-Transformationen).
- Entlastet Publisher von Aufgaben des Notification Producers.

Ausgehend von den oben beschriebenen Unterschieden und den daraus resultierenden Vorteilen von WS-BrokeredNotification lassen sich die Einsatzgebiete von WS-BaseNotification und WS-BrokeredNotification folgendermaßen abgrenzen (vgl. [Vinoski (2004a)]):

- WS-BaseNotification ist nur in abgeschlossenen Systemen sinnvoll.
- WS-BaseNotification ist nur für kleine Anzahlen von Producern und Consumern sinnvoll.
- WS-BaseNotification ist nur sinnvoll, wenn Kopplung zwischen Consumern und Producern vernachlässigbar ist.
- WS-BrokeredNotification bietet hohe Skalierbarkeit.
- WS-BrokeredNotification eliminiert Kopplung zwischen Consumern und Producern.

2.3.3 WS-Eventing

Mit WS-Eventing (vgl. [Box u. a. (2004b)]) existiert eine weitere Spezifikation, die ereignisbasierte Kommunikation in der Web-Service Welt zum Ziel hat. Allerdings ist WS-Eventing eine proprietäre Spezifikation eines Firmenkonsortiums und kein Industriestandard, wie das durch die OASIS standardisierte WS-Notification. WS-Eventing weißt einige Gemeinsamkeiten mit den WS-Notification Spezifikationen auf, geht jedoch weniger in die Tiefe und deckt etwa den Umfang von WS-BaseNotification ab (vgl. [Vinoski (2004a)]). Beide Spezifikationen definieren Schnittstellen für Subscription und Nachrichtenzustellung und basieren auf WS-Addressing. Im Gegensatz zu WS-Notification verwendet WS-Eventing keine Topics zur Nachrichtenfilterung sondern verwendet hierfür allgemeine Filterausdrücke, mit denen sich jedoch ein Topic-Konzept realisieren lässt.

WS-Eventing bietet keine Integration in WSDL, was dazu führt, dass es nur sehr schwer möglich ist jenseits von SOAP über HTTP andere Bindings zu benutzen (vgl. [Vinoski (2004b)]).

Im Gegensatz zu WS-Notification definiert WS-Eventing kein eigenes Nachrichtenformat, das als Envelope für Nachrichten verwendet werden kann, sondern überlässt den Aufbau der Benachrichtigungen vollständig der Anwendung.

Insgesamt kann man WS-Eventing als Teilmenge von WS-Notification ansehen und es bleibt abzuwarten, ob die beiden konkurrierenden Spezifikationen nicht zusammengefasst werden (vgl. [Vinoski (2004a)]).

2.4 WS-Policy

Die von WSDL gebotenen Möglichkeiten der syntaktischen Beschreibung eines Web-Service reichen nicht aus, um die Anforderungen, Fähigkeiten und Zusicherungen (zusammengefasst als *Policies*, im weiteren Sinne Nutzungsrichtlinien, bezeichnet) eines Web-Service

zu beschreiben. Das *Web Service Policy Framework* definiert zu diesem Zweck eine Möglichkeit die nicht-funktionalen Eigenschaften eines Web-Service zu beschreiben. Es wird nur eine Grammatik zur Beschreibung dieser Eigenschaften definiert, nicht jedoch das domänenspezifische Vokabular, mit dem die Eigenschaften beschrieben werden, das je nach Verwendungszweck spezifisch festgelegt werden muss.

Die Trennung zwischen dem Web Service Policy Framework und WSDL ist sinnvoll, da Policies i. d. R. eine andere Granularität als WSDL-Dokumente haben, potentiell wiederverwendbar sind und zu verschiedenen Zeitpunkten im Lebenszyklus eines Web-Service hinzugefügt werden können (vgl. [Ortiz und Leymann (2006)]).

Die Familie der Web Services Policy Framework Spezifikationen besteht aus drei Teilen. Das Web Services Policy Framework (vgl. [Bajaj u. a. (2006b)]), kurz WS-Policy genannt, beschreibt die Grammatik mit der Policies beschrieben werden und Mechanismen um Policies zu verarbeiten. Die Web Services Policy Attachment Spezifikation (vgl. [Bajaj u. a. (2006a)]), abgekürzt WS-PolicyAttachment, beschreibt Mechanismen, wie Policies mit verschiedensten Entitäten, wie z. B. WSDL-Dokumenten, XML-Elementen, UDDI-Einträgen und Web-Service Endpunkten, verbunden werden können. Die Web Services Policy Assertions Language (vgl. [Box u. a. (2002)]), oder kurz WS-PolicyAssertions, beschreibt schließlich einige Policy-Assertions, mit denen grundlegende, mit Messaging verbundene, Eigenschaften in WS-Policy beschrieben werden können.

Im Folgenden werden die Grundlagen von WS-Policy und WS-PolicyAttachment näher beschrieben. Aus WS-PolicyAssertions wird nicht näher eingegangen, da es für diese Arbeit keine Bedeutung hat.

2.4.1 Web Services Policy Framework

Das WS-Policy Framework beschreibt ein abstraktes Modell, mit dem Policies beschrieben werden können. Dieses abstrakte Modell definiert die Bausteine einer Policy und ihre Beziehungen. Die folgenden drei logischen Einheiten werden definiert:

- Policy Assertion
- Policy Alternative
- Policy

Abbildung 2.6 zeigt, die hierarchischen Beziehungen der logischen Einheiten zueinander.

Zusätzlich definiert WS-Policy auch Operationen, mit deren Hilfe aus mehreren Policies die effektive Policy abgeleitet werden kann, d. h. einen Mechanismus, wie sich mehrere Partner

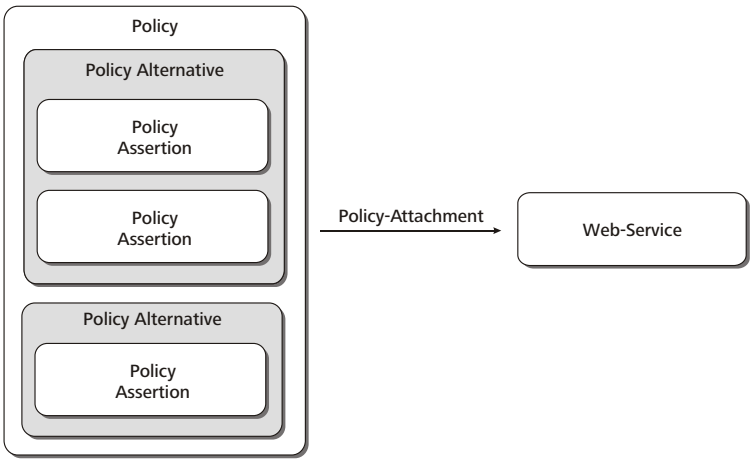


Abbildung 2.6: Überblick über WS-Policy

auf eine gemeinsame Policy einigen können (oder feststellen, dass es keine gemeinsame Policy gibt). Diese Operationen sind für diese Arbeit nicht von Belang, weshalb sie nicht näher erläutert werden.

2.4.1.1 Policy Assertions

Das Grundelement einer WS-Policy ist die *Policy Assertion* (vgl. [Bajaj u. a. (2006b)]). Sie beschreibt eine domänenspezifische Semantik, die in einer separaten, domänenspezifischen Spezifikation definiert ist. Policy Assertions sind typgebunden, wobei der Typ aus dem qualifizierten Namen (QName) der Assertion definiert wird. Policy Assertions sind also Instanzen eines Policy Assertion Typs, der ein Schema für die Instanzen impliziert. Eine Policy Assertion kann Kind-Elemente und Attribute enthalten, diese dienen jedoch nur näheren Beschreibung und ändern nichts am Typ der Policy Assertion.

Es ist möglich, Policy Expressions in eine Policy Assertion einzuschachteln, um komplexere Anforderungen oder Eigenschaften auszudrücken oder die umschließende Policy Assertion näher zu bestimmen.

2.4.1.2 Policy Alternatives

Eine *Policy Alternative* (vgl. [Bajaj u. a. (2006b)]) ist ein logisches Konstrukt, das eine Menge von Policy Assertions repräsentiert. Hierbei definiert die Policy Alternative keine Ordnung über den enthaltenen Policy Assertions, dies bleibt der domänenspezifischen Verarbeitung der enthaltenen Assertions überlassen.

Die Semantik einer Policy Alternative wird allein durch die Semantiken der enthaltenen Policy Assertions bestimmt, wobei eine Entität eine Policy Alternative genau dann unterstützt, wenn sie alle in der Alternative enthaltenen Assertions unterstützt. Die Menge aller Policy Assertion Typen in einer Policy Alternative wird als *Vokabular* der Policy Alternative bezeichnet. Die Abwesenheit eines Policy Assertion Typs in einer Alternative bedeutet hierbei, dass die Policy Alternative Zusicherungen dieses Typs explizit ausschließt.

2.4.1.3 Policies

Eine *Policy* (vgl. [Bajaj u. a. (2006b)]) ist eine Menge von Policy Alternatives, wobei die Policy, die keine Policy Alternatives enthält als *leere Policy* bezeichnet wird. Eine leere Policy enthält keine Auswahlmöglichkeiten und macht auch keinerlei Aussagen.

Analog zum Vokabular einer Policy Alternative wird auch das *Vokabular* einer Policy als die Vereinigung der Vokabulare der enthaltenen Policy Alternatives definiert.

Die Semantik einer Policy ist so definiert, dass eine Entität eine Policy genau dann unterstützt, wenn sie mindestens einer der in der Policy enthaltenen Policy Alternativen unterstützt.

2.4.1.4 Policy Expressions

Eine *Policy Expression* (vgl. [Bajaj u. a. (2006b)]) ist eine Repräsentation einer Policy als XML-Infoset. Eine Policy Expression kann mit einem Namen versehen werden, über den sie referenziert werden kann. Eine Policy Expression kann sog. *Policy Operatoren* enthalten. Es gibt zwei Policy Operatoren: `<wsp:All>` (synonym hierzu kann auch `<wsp:Policy>` verwendet werden) `<wsp:ExactlyOne>` und der `Optional`-Operator. Die Policy Operation `<wsp:All>` beschreibt eine Policy Alternative, die aus allen im All-Element geschachtelten Policy Assertions besteht. Die Operation `<wsp:ExactlyOne>` beschreibt, dass genau eine der enthaltenen Policy Assertions oder Operatoren gleichzeitig an einer Alternative teilhaben kann. Der `Optional`-Operator wird als Attribut in eine Policy Assertion eingebracht, deren Betrachtung für die Policy optional ist, also ignoriert werden kann (vgl. [Weerawarana u. a. (2005)]).

Listing 2.1: Beispiel für eine WS-Policy (entnommen aus [Bajaj u. a. (2006b)])

```
1 <wsp:Policy
2   xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
3   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
4   <wsp:ExactlyOne>
5     <wsp:All>
6       <sp:Basic256Rsa15 />
```

```
7       </wsp:All>
8       <wsp:All>
9         <sp:TripleDesRsa15 />
10      </wsp:All>
11    </wsp:ExactlyOne>
12  </wsp:Policy>
```

2.4.2 Web Services Policy Attachment

WS-PolicyAttachment beschreibt verschiedene Mechanismen um Policies mit Subjekten zu verknüpfen. Policies allein sind nicht in der Lage zu Beschreiben für welche Subjekte sie gelten sollen. Durch diese Trennung wird es möglich, Policies wiederzuverwenden und mehreren verschiedenen Subjekten zuzuweisen. Subjekte können in diesem Kontext beispielsweise WSDL-Dokumente, ein Nachrichtenaustausch, ein Web-Service Endpunkt oder ein Porttyp etc. sein (vgl. [Weerawarana u. a. (2005)]). Abbildung 2.6 illustriert dies beispielhaft für einen Web-Service Endpunkt.

Die WS-PolicyAttachment Spezifikation beschreibt zwei grundlegende Mechanismen, wie Policies mit Subjekten verbunden werden können (vgl. [Bajaj u. a. (2006a)]):

XML Element Attachments Hierbei werden die Policy Attachments direkt in die XML-Elemente, die die Subjekte beschreiben, eingebettet. Die Zuordnung von Policies zu einem XML-Element kann hierbei auf zwei Arten erfolgen:

- Über ein Attribut, das die URIs der anzuwendenden Policies referenziert.
- Durch ein dediziertes Kind-Element, das die Policies referenziert.

Externe Policy Attachments Hierbei erfolgt die Zuordnung der Policy zu einem Subjekt außerhalb der eigentlichen Definition des Subjekts. Zu diesem Zweck definiert WS-PolicyAttachment einen Container, der eine oder mehrere Policies enthalten oder referenzieren kann und diese mit einem oder mehreren über eine sog. *Domain Expression* referenzierten Subjekten verbinden. Die Form einer Domain-Expression ist domänenspezifisch und wird von WS-PolicyAttachment nicht näher spezifiziert. Ein Beispiel für eine solche Domain-Expression ist eine WS-Addressing Endpunkt-Referenz, die einen Web-Service Endpunkt referenziert, für den eine Policy gelten soll. Dieser Mechanismus ist v. a. dann von Bedeutung, wenn eine Policy nachträglich hinzugefügt werden soll, oder keine XML-Beschreibung des Subjekts vorliegt oder diese nicht modifiziert werden kann oder soll.

Die WS-PolicyAttachment Spezifikation beschreibt des weiteren, wie WS-Policies an WSDL-Dokumente, UDDI-Einträge etc. angehängt werden können und wie aus solchen Dokumenten die effektive Policy bestimmt werden kann. Dies ist für diese Arbeit nicht von Belang, deshalb sei für Details hierzu auf die Spezifikation (vgl. [Bajaj u. a. (2006a)]) verwiesen.

KAPITEL 3

Anforderungen und Konzeption

Dieses Kapitel beschreibt die konkreten Anforderungen an eine Lösung der Aufgabenstellung, die in Abschnitt 1.3 vorgestellt wurde und skizziert ein Konzept zur Lösung dieser Fragestellung. Das Konzept wird hier technologieunabhängig beschrieben, seine Umsetzung wird dann in Kapitel 4 detailliert erläutert.

3.1 Anforderungen

Im Folgenden werden die Anforderungen, die an eine AOP-fähige BPEL-Engine und einen Aspect-Weaver für BPEL-Prozesse zu stellen sind, beschrieben.

3.1.1 Anforderungen an Definition und Verwaltung der Aspekte

Die Definition eines *Aspekts* orientiert sich an der Terminologie von *AspectJ* (siehe Abschnitt 2.2.5), adaptiert sie jedoch für die Anwendung auf BPEL-Prozesse (vgl. [Karastoyanova u. a. (2007)]). Ein Aspekt soll durch eine WS-Policy dargestellt werden, die Zuordnung (Deployment) eines Aspekts zu einem Prozess oder einer Prozessinstanz soll durch einen WS-PolicyAttachment Mechanismus erfolgen. Ein Aspekt soll dabei aus einem *Advice* und einem oder mehreren *Pointcuts* bestehen. Ein Advice besteht wiederum aus einer Spezifikation, wann er ausgeführt werden soll, einem *Web-Service Port* (hierfür soll eine WS-Addressing Endpunktreferenz (vgl. [Box u. a. (2004a)]) verwendet werden), einer *Operation*, die aufgerufen werden soll und optional *Transformationen* der Parameter der Operation. Die Spezifikation eines *Pointcuts* identifiziert eine oder mehrere Stellen innerhalb eines BPEL-Prozesses. Die Zusammenhänge sind nochmals in Abbildung 3.1 dargestellt.

Darüber hinaus soll es möglich sein, Aspekte in sog. *Aspektgruppen* zusammenzufassen. Eine

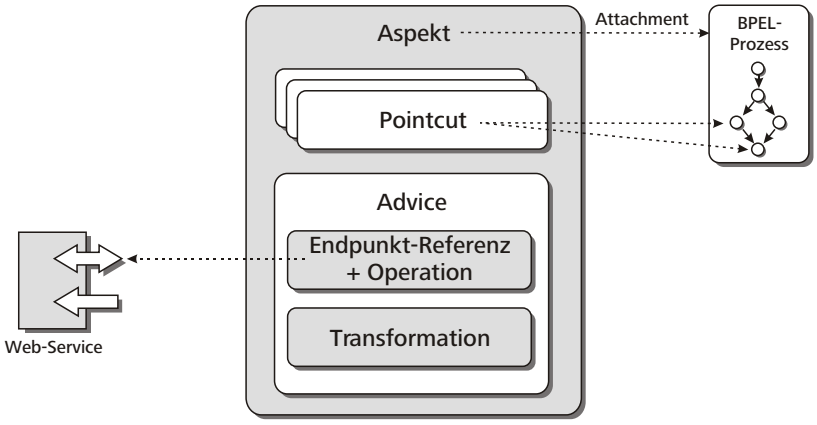


Abbildung 3.1: Struktur eines Aspekts

Aspektgruppe bildet eine logische Einheit zur Gruppierung von Aspekten und definiert, dass die enthaltenen Aspekte nur alle gemeinsam deployt und angewendet werden dürfen (in Abschnitt 3.1.2 findet sich eine nähere Beschreibung dieser Funktionalität).

An die Verwaltung der Aspekte werden folgende funktionale Anforderungen gestellt:

- Es muss möglich sein Aspekte anzulegen, zu bearbeiten und zu löschen.
- Es muss möglich sein, einen Aspekt auszuwählen und ihn zu deployen, wobei ausgewählt werden kann, für welchen Prozess oder für welche Prozessinstanz (identifiziert über die Instanz-ID) dieser Aspekt gelten soll.
- Es muss möglich sein, einen deployten Aspekt auszuwählen und in zu undeployen.
- Es muss angezeigt werden, welche Aspekte momentan existieren.
- Es muss angezeigt werden, welche Aspekte momentan für welche Prozesse oder Instanzen deployt sind.
- Die definierten Aspekte sollen persistent abgelegt werden.

3.1.2 Anforderungen an den Aspect-Weaver

Der Aspekt-Weaver muss sich mit einer BPEL-Engine integrieren. Hierzu ist es erforderlich, dass der Quellcode der BPEL-Engine verfügbar und gut wartbar ist, damit die hierfür notwendigen Änderungen an der Engine vorgenommen werden können.

Der Aspekt-Weaver soll mindestens die folgenden Advice-Typen unterstützen, um eine sinnvolle Adaption von BPEL-Prozessen zu erlauben:

before Advice wird *vor* den durch den Pointcut selektierten Aktivitäten ausgeführt.

after Advice wird *nach* den durch den Pointcut selektierten Aktivitäten ausgeführt.

instead Advice wird *anstelle* der durch den Pointcut selektierten Aktivitäten ausgeführt.

Die folgenden Joinpoints sollen unterstützt werden:

Aktivitäten Advice kann die Aktivität um neue Funktionalitäten ergänzen oder die Aktivität komplett entfernen.

Links Ein Advice kann die Transition-Condition eines Links ändern, falls beispielsweise temporär eine komplexere Regel erforderlich wird.

Der Aufruf der Advices soll über WS-Notification erfolgen, wobei jedem Pointcut ein eigener Topic zugeordnet wird.

Es ist erforderlich, dass der Aspect-Weaver eine Schnittstelle für das Deployment von Aspekten bietet. Die deployten Aspekte sollen bis zum Undeployment persistent gespeichert werden, um Neustarts des Weavers zu überstehen.

Der Weaver muss mit den Joinpoints verbundene Variablen aus dem Prozess verwenden können, um sie als Parameter für die einzuwebenden Web-Services zu nutzen. Der Mechanismus, wie und welche Variablen an einem Joinpoint verwendet werden, ist zu definieren. Es soll eine Möglichkeit zur Transformation der Parameter geben, um eine Anpassung an abweichende Schnittstellen eines einzuwebenden Web-Service zu erlauben.

Zur Definition der Pointcuts sollen XPath-Ausdrücke verwendet werden, die auf den BPEL-Prozess angewendet werden.

3.2 Konzeption

Im Folgenden wird die Konzeption zur Realisierung der oben dargestellten Anforderungen vorgestellt. Dabei wird zunächst auf die Architektur des Gesamtsystems eingegangen. Anschließend wird das Konzept für die Anbindung des Aspekt-Weavers an die BPEL-Engine beschrieben. Im Anschluss daran wird die detaillierte Konzeption der einzelnen Bestandteile der Architektur näher betrachtet.

3.2.1 Vorüberlegungen

Für die flexible Adaption von langlaufenden Prozessen ist es unabdingbar, eine Möglichkeit zur Adaption zur Laufzeit zu haben. Aus diesem Grund ist es erforderlich, dass der Aspekt-Weaver dynamisches Weaving unterstützt. Diesen Punkt gilt es bei der Konzeption der Systemarchitektur zu beachten.

Bevor es möglich ist, sich auf eine bestimmte Systemarchitektur festzulegen, ist es erforderlich, den eigentlichen Weaving-Mechanismus festzulegen. Insbesondere muss ein grundsätzliches Vorgehen definiert werden, das genutzt wird, um den Aspekt-Weaver an eine BPEL-Engine anzubinden. Hierfür existieren mehrere Ansätze, die alle ihre Vor- und Nachteile haben:

Präprozessor Dieser Ansatz nutzt einen Präprozessor um die Aspekte mit den BPEL-Prozessen fest zu verweben. Dies kann spätestens zur Deployment-Zeit erfolgen. Eine Anpassung zur Laufzeit ist nicht möglich. Der Vorteil dieses Ansatzes ist, dass potentiell keine Änderungen an einer BPEL-Engine notwendig sind und dass er mit jeder BPEL-Engine funktioniert. Die Nachteile sind u. a. sehr geringe Flexibilität und die Tatsache, dass dynamisches AOP nicht unterstützt werden kann. Ebenso ist keine instanzbasierte Adaption (oder nur mit sehr viel Aufwand im Sinne von komplexen Änderungen des Kontrollflusses) möglich. Dieser Ansatz findet sich beispielsweise bei [Baresi und Guinea (2005)].

Direkte Integration Dieser Ansatz geht von einer bestehenden BPEL-Engine aus und fügt dieser neuen Code hinzu, der den enthaltenen BPEL-Interpreter befähigt direkt Aspekte auszuführen. Er wurde beispielsweise von [Erradi u. a. (2005)] verfolgt.

Dieser Ansatz ist vom Aufwand her vermutlich der am wenigsten aufwändige. Er ist jedoch auch mit einigen Nachteilen behaftet. Beispielsweise bindet man sich an eine bestimmte Version einer BPEL-Engine (was bei Eigenentwicklungen kein Problem sein muss), so dass das Nachziehen von Updates ein potentielles Problem darstellt.

Neuentwicklung Bei diesem Ansatz wird eine BPEL-Engine mit Aspektunterstützung von Grund auf neu konzipiert und implementiert. [Courbis und Finkelstein (2004, 2005a)] nutzen diesen Ansatz, um unter Verwendung des von ihnen entwickelten Werkzeugs „SmartTools“ eine neue BPEL-Engine mit AOP-Unterstützung zu realisieren. Dies hat den Vorteil, dass man eine sehr gute Integration des AOP-Teils in die BPEL-Engine erreichen kann. Dem gegenüber steht ein sehr hoher Aufwand für die vollständige Entwicklung einer neuen BPEL-Engine.

Event Publishing Dieser Ansatz basiert drauf, die Zustandsübergänge der einzelnen Elemente eines Prozesses während der Navigation in Form von Ereignissen zugänglich zu machen. [Cilia u. a. (2003)] beschreiben, dass man die Reaktion auf Ereignisse, im Sinne von *ECA-Regeln* (Event-Condition-Action-Regeln aus der Welt der reaktiven Datenbanksysteme), auf aspektorientierte Programmierung abbilden kann. Hierbei

werden die Pointcuts auf Ereignisse (oder eine Kombination aus Ereignis und Quelle) abgebildet und die Advices als Action aufgefasst.

Dieser Ansatz hat den Vorteil, dass nur kleine Änderungen an der eigentlichen BPEL-Engine vorgenommen werden müssen und der Weaver unabhängig von der Engine entwickelt werden kann, vorausgesetzt es liegt ein nicht von der Engine abhängiges Event-Modell vor. Ein Nachteil dieses Ansatzes ist die potentiell geringere Performanz, wenn der Weaver als externe Komponente realisiert wird und nicht in die Engine integriert.

Ähnliche Ansätze wurden auch bereits für die Realisierung einer verteilten Workflow-Engine verwendet (vgl. [Geppert und Tombros (1998); Jablonski u. a. (1995)]).

Für diese Arbeit wird der Ansatz des Event Publishing gewählt, da er sich sehr gut mit den von [Karastoyanova u. a. (2007)] beschriebenen Ansätzen deckt, weicht jedoch insofern von diesem ab, als dass hier die Events der Engine noch von einem Weaver verarbeitet werden müssen und nicht direkt als Notification den Aufruf des Advice auslösen.

3.2.2 Systemarchitektur

Ausgehend von dem Weaving-Ansatz (Abschnitt 3.2.1) wurde die konzeptionelle Architektur für das Gesamtsystem entworfen (siehe Abbildung 3.2).

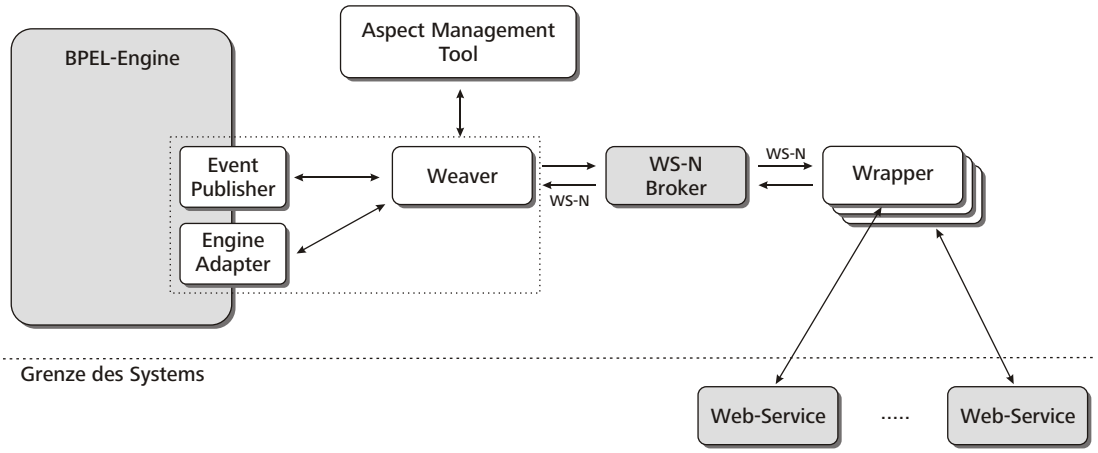


Abbildung 3.2: Konzeptionelle Architektur des Systems (grau hinterlegte Komponenten sind nicht Bestandteil dieser Arbeit, die Abkürzung „WS-N“ steht für WS-Notification)

Der *Event Publisher* ist die Erweiterung der BPEL-Engine, die die Navigationsevents nach außen verfügbar macht. Sie informiert den Weaver über die Navigationsevents und empfängt die Resultate der eingewobenen Aspekte und führt sie in den Prozess zurück.

Der *Engine-Adapter* ist eine zusätzliche Schnittstelle, über die der Weaver mit der Engine kommuniziert. Er dient beispielsweise dazu, dem Weaver Zugriff auf Variableninhalte oder die Prozessmodelle der deployten Prozesse zu geben.

Die zentrale Komponente der Architektur bildet der *Weaver*. Er ist für die Identifikation der einzuwebenden Aspekte, die Verwaltung der deployten Aspekte und den Versand der Notification Nachrichten an den oder die Wrapper zuständig. Er agiert als Broker (vgl. [Buschmann u. a. (1996)]) zwischen dem Event Publisher und dem Wrapper bzw. den einzuwebenden Web-Services. Wenn man die Architektur abstrakter betrachtet, kann man auch den Weaver, den Event Publisher und den Engine-Adapter zusammenfassen (in Abbildung 3.2 durch den gepunkteten Rahmen dargestellt) und diese drei Komponenten zusammen als *Aspekt-Weaver im weiteren Sinne* betrachten. Durch diese Abstraktion entspricht das hier beschriebene Konzept dem von [Karastoyanova u. a. (2007)] vorgestellten.

Der *Wrapper* bildet die Komponente, die die Aufrufe der einzuwebenden Services übernimmt und die Resultate an den Weaver zurückgibt. Es ist nicht zwingend, dass nur ein Wrapper innerhalb des Systems existiert, der alle Aufrufe übernimmt. Es ist ebenso möglich, dass eine Partitionierung der Pointcuts vorgenommen wird und Partitionen sowie die zugehörigen Web-Services verschiedenen Wrappern zugeteilt werden. Die im Rahmen dieser Arbeit erstellte Realisierung beschränkt sich jedoch auf einen einzigen Wrapper, der alle Web-Service Aufrufe abwickelt (siehe Abschnitt 4.7).

Die Verwaltung und das Deployment der Aspekte übernimmt das *Aspekt Management Tool*. Es ist eine graphische Anwendung, die der Benutzer verwendet um Aspekte anzulegen und zu bearbeiten, sowie sie zu deployen und undeployen. Das Aspekt Management Tool besitzt zum Durchführen des Deployments eine Schnittstelle zum Weaver. Weitere Details werden in Abschnitt 3.2.6 dargestellt.

3.2.3 Anbindung an die Engine

Die Anbindung an die BPEL-Engine ist zweigeteilt. Zum einen wird durch den *Event Publisher* eine Komponente eingesetzt, die einzig und allein für die Verteilung von Events für Statusübergänge während der Navigation durch einen Prozess zuständig ist. Zusätzlich gibt es den *Engine-Adapter*, der dem Zugriff auf die Interna der Engine dient.

3.2.3.1 Event Publishing

Für die Verteilung von Ereignissen, die während der Navigation durch einen Prozess entstehen, ist zunächst einmal ein generisches Event-Modell für BPEL-Prozesse notwendig, das

von den internen, implementierungspezifischen Zuständen und Events einer konkreten BPEL-Engine abstrahiert. Ein solches Event-Modell ist nicht nur für die Nutzung zur AOP-Unterstützung sinnvoll, sondern kann für diverse andere Zwecke wie Koordination von verteilten Prozessen, Transaktionssteuerung etc. von Vorteil sein. Aus diesem Grund wurde in [Khalaf u. a. (2007)] ein solches generisches Event-Modell entwickelt und einige Anwendungsfälle vorgestellt. Das Modell bezieht sich auf die Spezifikation von BPEL Version 1.1, sollte jedoch auf die kommende Version 2.0 mit kleinen Erweiterungen übertragbar sein.

Um den Ablauf des BPEL-Prozesses von außen steuern zu können, wurden einige Event-Typen als potentiell blockierend definiert. Hierdurch wird es möglich, die Engine mit einer externen Verarbeitung der Events zu synchronisieren. Zum Auflösen einer Blockierung wurden sog. Incoming Events definiert, die zusätzlich weitere Aktionen auslösen können und von externen Quellen stammen. Die Definition, welche Events wirklich blockieren sollen, erfolgt auf Prozessmodellebene.

Im Folgenden wird das Event-Modell und das zugrundeliegende Zustandsmodell vorgestellt.

3.2.3.2 Event-Modell für Aktivitäten

Abbildung 3.3 zeigt das Zustandsmodell für allgemeine BPEL Aktivitäten.

Die einzelnen Events, die im Lebenszyklus einer Aktivität auftreten können, werden im Folgenden näher beschrieben.

Activity_Ready Dieses Event wird ausgelöst, wenn eine Aktivität bereit zur Ausführung ist, d. h. wenn ihre eingehenden Links ausgewertet sind und ihre Join-Condition wahr ist. Das Event ist potentiell blockierend und kann durch das Incoming Event „Start_Activity“ fortgesetzt werden.

Activity_Executing Dieses Event wird gefeuert, wenn eine Aktivität ihre eigentliche Ausführung beginnt. Für Aktivitäten wie `<pick>`, `<receive>` etc. oder `<onMessage>`-Handler wird dieses Event ausgelöst, wenn sie beginnen auf Nachrichten zu warten und nicht erst, wenn die eigentliche Nachricht empfangen wird.

Activity_Executed Das Event „Activity_Executed“ wird ausgelöst, wenn eine Aktivität ihre Ausführung beendet hat, dieses aber noch nicht als abgeschlossen markiert ist. Wenn das Event als blockierend markiert ist, geht die Engine in den „Waiting“-Zustand über und wartet auf das Incoming Event „Complete_Activity“.

Activity_Complete Dieses Event wird ausgelöst, wenn eine im Zustand „Waiting“ blockierte Aktivität das Incoming Event „Complete_Activity“ empfangen hat oder eine Aktivität erfolgreich ausgeführt wurde und kein Warten notwendig ist.

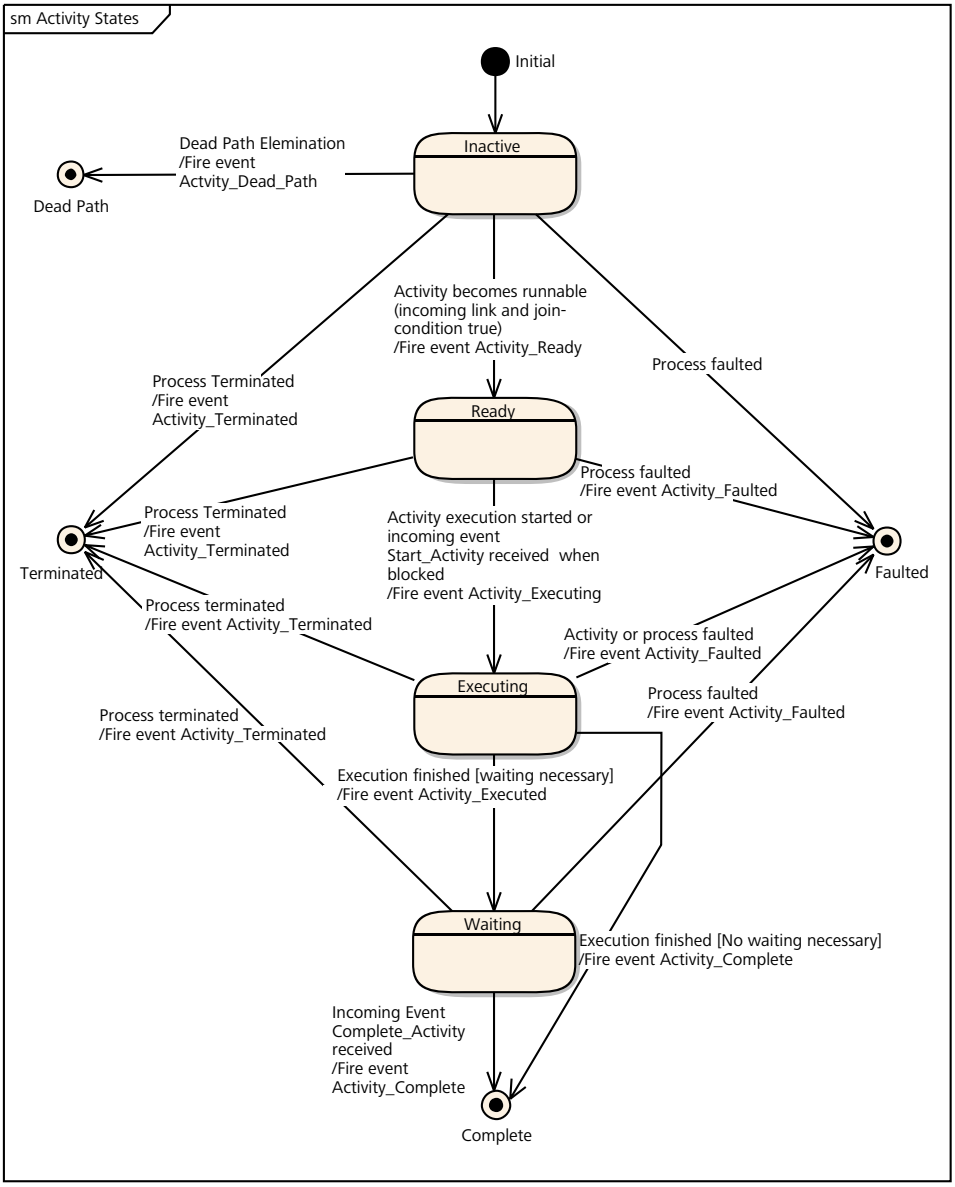


Abbildung 3.3: Zustandsdiagramm für allgemeine Aktivitäten

Activity_Dead_Path Diese Event wird ausgelöst, wenn eine Aktivität durch die Dead-Path-Elimination der Engine als zum Dead-Path gehörig markiert wurde, d. h. die Join-Condition zu „false“ ausgewertet wurde.

Activity_Terminated Dieses Event wird ausgelöst, wenn eine Aktivität als terminiert oder abgebrochen markiert wird, weil die Engine die zugehörige Prozessinstanz terminiert.

Activity_Faulted Dieses Event wird ausgelöst, wenn eine Aktivität abgebrochen wird, weil innerhalb der Aktivität eine Ausnahme aufgetreten ist. Sie wird ebenfalls ausgelöst, wenn eine Aktivität übersprungen wird, weil in einer vorhergehenden Aktivität desselben Scopes eine nicht behandelte Ausnahme aufgetreten ist.

3.2.3.3 Event-Modell für Scopes

Scopes sind eine spezielle Art von strukturierten Aktivitäten, die zur Behandlung von Ereignissen (hierbei sind prozess-interne Ereignisse, wie der Eingang einer Nachricht oder das Auslösen eines zeitgesteuerten Alarms gemeint, nicht die in diesem Abschnitt vorgestellten Navigations-Events) und Ausnahmen im Kontrollfluss dienen. Durch diese erweiterten Eigenschaften ist ein abweichendes, mächtigeres Event-Modell für Scopes notwendig. In Abbildung 3.4 ist dieses dargestellt. Im Folgenden werden die in diesem Modell neu hinzugekommenen Events beschrieben.

Scope_Handling_Fault Dieses Event wird ausgelöst, wenn der Fault-Handler (hierbei ist es unbedeutend, ob dies ein expliziter oder der implizite Fault-Handler ist) eines Scopes durch eine Ausnahme aktiviert wird.

Scope_Complete_With_Fault Dieses Event wird ausgelöst, wenn ein Scope als komplett markiert wird, nachdem eine Ausnahme behandelt und nicht durch den Fault-Handler zum umschließenden Scope propagiert wurde.

Scope_Handling_Event Dieses Event wird ausgelöst, wenn der Event-Handler (egal ob onAlarm oder onMessage) eines Scopes ausgeführt wird.

Scope_Event_Handling_Ended Dieses Event wird ausgelöst, wenn die Ereignisbehandlung im Event-Handler eines Scopes abgeschlossen wurde.

Scope_Compensating Dieses Event wird ausgelöst, wenn der Compensation-Handler eines Scopes durch die Engine oder durch das Incoming Event „Compensate_Scope“ ausgelöst wurde.

Scope_Compensated Dieses Event wird ausgelöst, wenn der Compensation-Handler eines Scopes abgeschlossen wird.

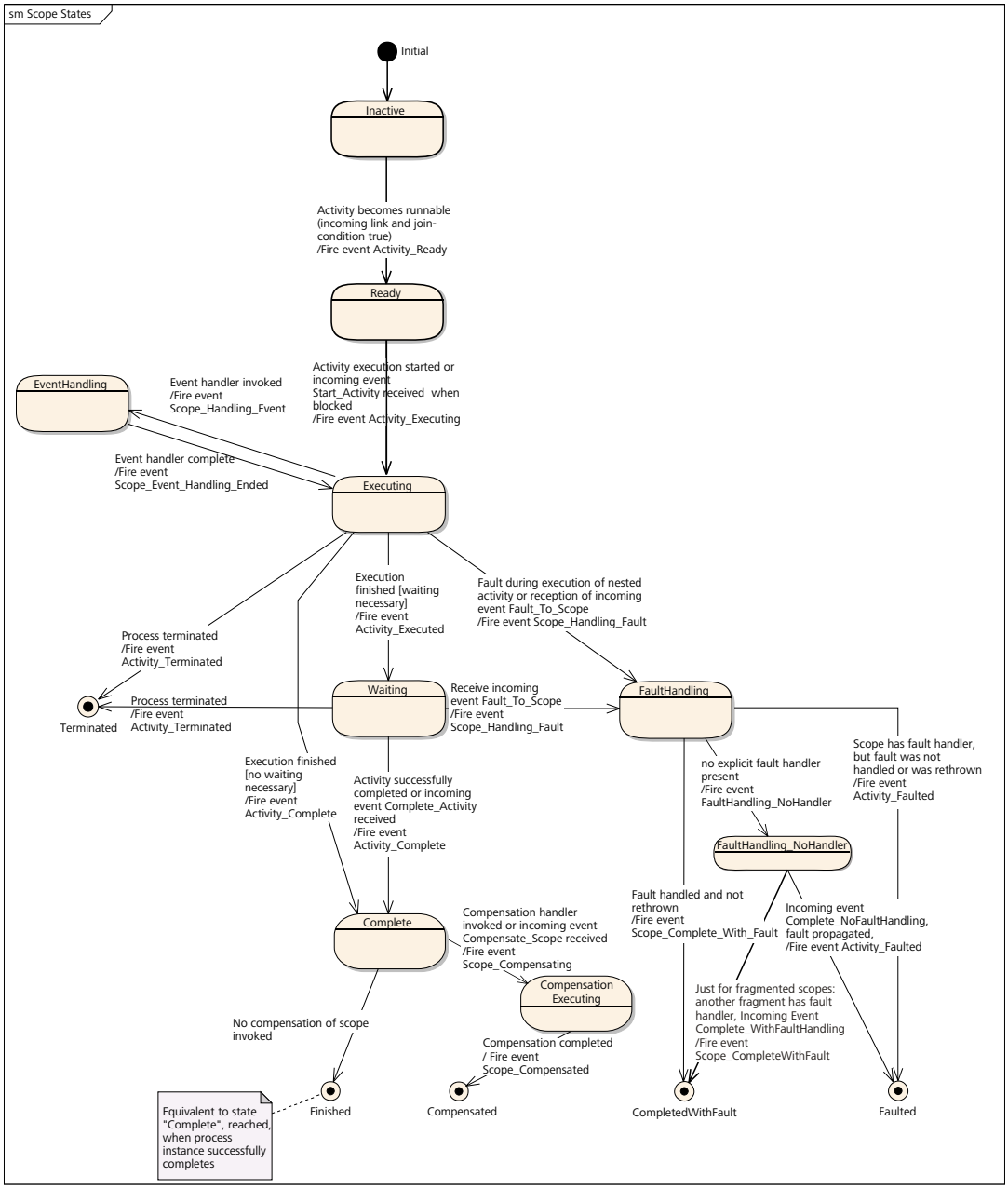


Abbildung 3.4: Zustandsdiagramm für Scopes (zur besseren Übersichtlichkeit sind einige Übergänge zu Terminated und Dead_Path weggelassen, diese können dem Zustandsdiagramm für Aktivitäten entnommen werden)

3.2.3.4 Event-Modell für Loops

Auch die Ausführung von Loops in BPEL-Prozessen weicht von der anderer Aktivitäten ab. In Abbildung 3.5 ist das zugehörige Zustandsdiagramm abgebildet.

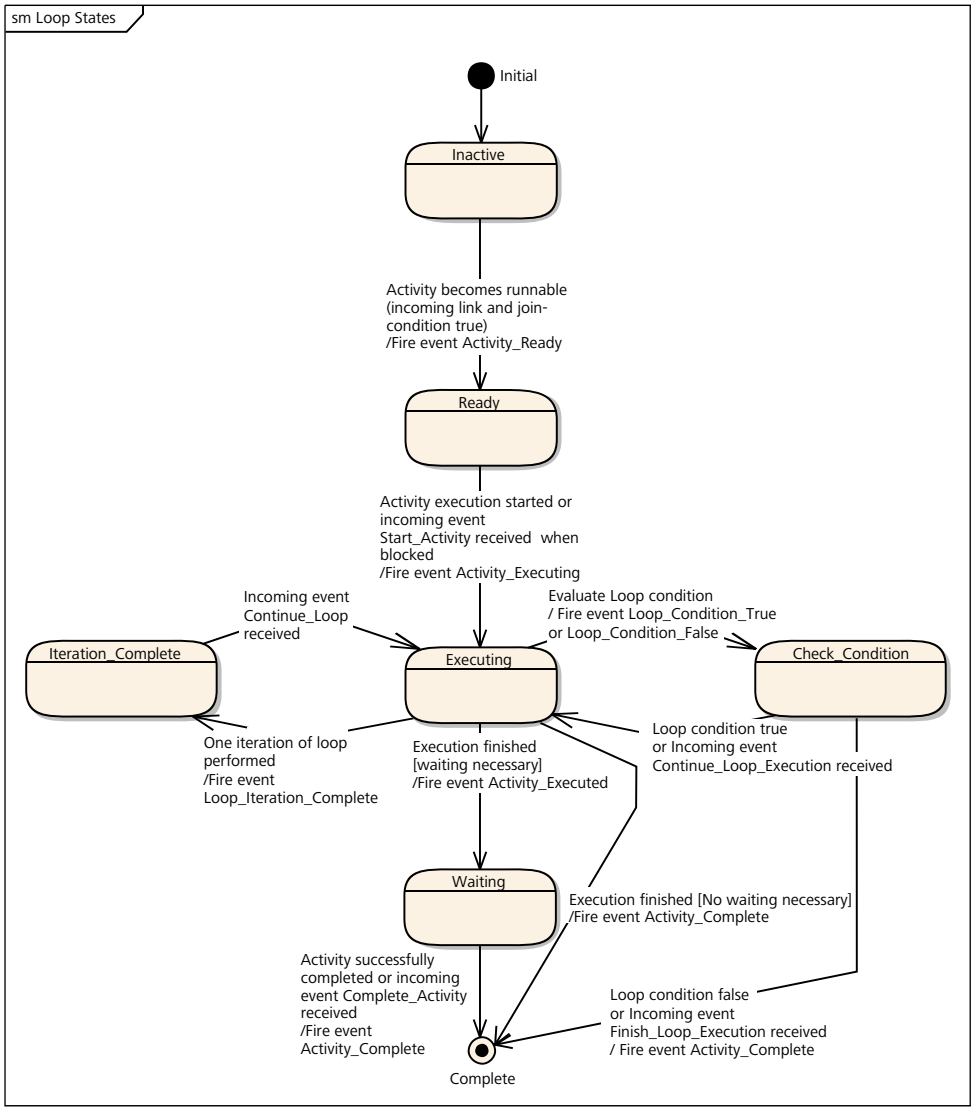


Abbildung 3.5: Zustandsdiagramm für Loops (zur besseren Übersichtlichkeit sind die Übergänge zu Terminated, Faulted und Dead_Path weggelassen, diese können dem Zustandsdiagramm für Aktivitäten entnommen werden)

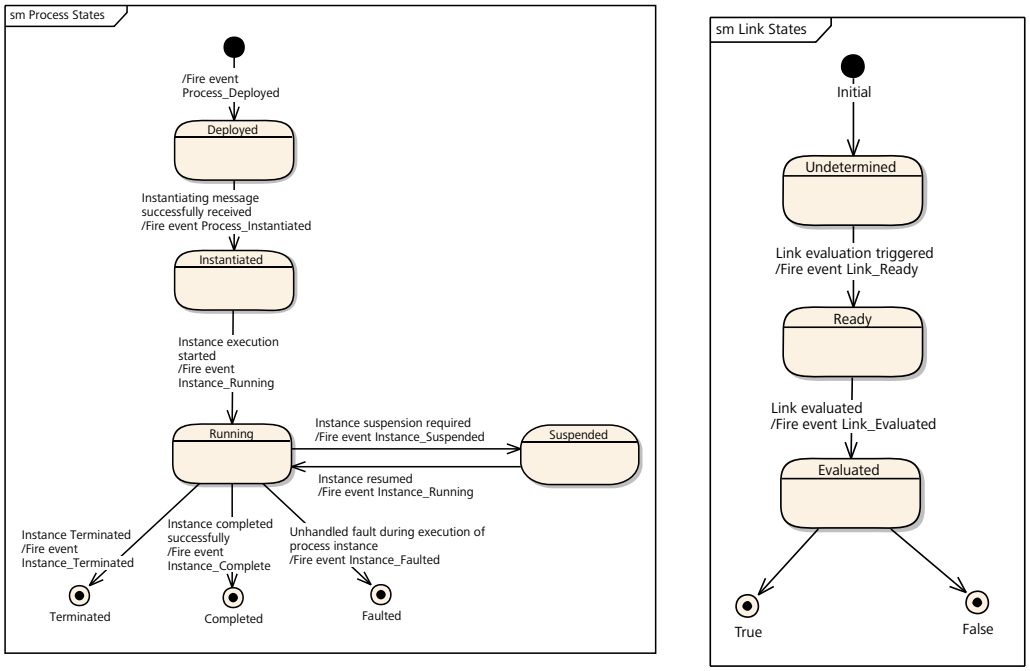
Loop_Condition_True Dieses Event wird ausgelöst, wenn die Bedingung eines Loops zu `true` ausgewertet wurde. Dieses Event ist vor allem dann von Bedeutung, wenn die Iteration von Loops mit anderen Aktivitäten synchronisiert werden soll.

Loop_Condition_False Dieses Event wird ausgelöst, wenn die Bedingung eines Loops zu `false` ausgewertet wurde. Dieses Event dient vornehmlich zur Synchronisation der Iteration von Loops mit anderen Aktivitäten.

Loop_Iteration_Complete Dieses Event wird ausgelöst, wenn eine Iteration eines Loops abgeschlossen ist. Der genaue Zeitpunkt seiner Auslösung ist vor der erneuten Auswertung der Schleifenbedingung. Dieses Event ist vor allem dann von Bedeutung, wenn die Iteration von Loops mit anderen Aktivitäten synchronisiert werden soll. Hierzu dient das Incoming Event „Continue_Loop“.

3.2.3.5 Event-Modell für Prozesse

Auch für den Lebenszyklus von Prozessen gibt es ein Event-Modell, das in Abbildung 3.6(a) dargestellt ist.



(a) Prozesse (b) Links

Abbildung 3.6: Zustandsdiagramme für Prozesse und Links

Folgende Events können während des Lebenszyklus einer Prozessinstanz auftreten:

Process_Deployed Dieses Event wird ausgelöst, wenn ein BPEL-Prozess in die Engine deployt wird. Es unterscheidet sich von den anderen Prozess-Events dadurch, dass es sich nicht auf eine Instanz, sondern auf einen ganzen Prozess bezieht.

Process_Instantiated Dieses Event wird durch die Instanziierung eines Prozesses ausgelöst, was durch den Empfang einer Nachricht durch eine `<pick>`- oder `<receive>`-Aktivität bzw. einen `<onMessage>`-Eventhandler mit dem Attribut `createInstance=yes` geschehen kann.

Instance_Running Dieses Event wird ausgelöst, wenn ein Prozess nach der Instanziierung zu laufen beginnt, oder wenn ein ausgesetzter Prozess fortgesetzt wird.

Instance_Suspended Dieses Event wird ausgelöst, wenn ein laufender Prozess ausgesetzt wird, z. B. indem ein Breakpoint erreicht wurde, oder von der Engine eine externe Anfrage die Instanz auszusetzen erhalten wurde.

Instance_Terminated Dieses Event wird ausgelöst, wenn eine Prozess-Instanz durch eine `<terminate>`-Aktivität terminiert wird.

Instance_Complete Dieses Event wird ausgelöst, wenn eine Prozessinstanz erfolgreich abgeschlossen wurde, also nicht durch einen Fehler oder eine `<terminate>`-Aktivität abgebrochen wurde.

Instance_Faulted Dieses Event wird ausgelöst, wenn eine Prozessinstanz durch eine Ausnahme abgebrochen wurde, die aus dem Fault-Handler des Prozesses (egal ob explizit oder implizit) heraus propagiert wurde.

3.2.3.6 Event-Modell für Links

Im Lebenszyklus von Links gibt es ebenfalls mehrere Zustände, deren Übergänge durch Events veröffentlicht werden. In Abbildung 3.6(b) sind die Zustände abgebildet.

Link_Ready Dieses Event wird ausgelöst, wenn ein Link bereit zum Auswerten ist, d. h. wenn seine Quellaktivität abgeschlossen ist.

Link_Evaluated Dieses Event wird ausgelöst, wenn die Transition-Condition eines Links ausgewertet wurde.

3.2.3.7 Event-Modell für Incoming Events

Um Aktionen innerhalb der Engine auszulösen, wurden die sog. *Incoming Events* definiert. Durch sie wird es auch ermöglicht, Aktivitäten, die blockiert sind und auf eine Antwort warten, fortzusetzen. Die folgenden Incoming Events sind definiert:

Compensate_Scope Dient dazu, einen Scope zu kompensieren, der sich im Zustand „Complete“ befindet.

Fault_To_Scope Dieses Incoming Event dient dazu, im Kontext eines Scopes, der sich im Zustand „Executing“ oder „Waiting“ befindet, einen Fault auszulösen. Der Fault wird hierbei durch die normale Hierarchie der Fault-Handler propagiert.

Start_Activity Dieses Event dient dazu, eine Aktivität, die im Zustand „Ready“ blockiert ist, fortzusetzen.

Complete_Activity Dieses Event dient dazu, eine im Zustand „Waiting“ blockierte Aktivität fortzusetzen und sie in den Zustand „Completed“ zu überführen. Zusätzlich kann dieses Event dazu genutzt werden, Aktivitäten, die im Zustand „Ready“ oder früher sind als abgeschlossen zu markieren, sie also zu überspringen (siehe auch Abschnitt 3.2.4.1).

Continue_Loop_Execution Dieses Event dient dazu, eine Loop-Aktivität, die im Zustand „Check_Condition“ blockiert ist, fortzusetzen und den Wert der Loop-Bedingung auf `true` zu setzen, also einen weiteren Durchlauf der Schleife zu erzwingen.

Finish_Loop_Execution Dieses Event dient dazu, eine Loop-Aktivität, die im Zustand „Check_Condition“ blockiert ist, fortzusetzen und den Wert der Loop-Bedingung auf `false` zu setzen, also die Verarbeitung der Schleife zu beenden.

Continue_Loop Dieses Event dient dazu, einen Loop, der am Ende eine Iteration im Zustand „Iteration_Complete“ blockiert ist, fortzusetzen und die erneute Auswertung der Schleifenbedingung anzustoßen.

Set_Link_State Dieses Event dient dazu, einen Link, der sich blockiert im Zustand „Evaluated“ befindet, auf einen neuen Wert zu setzen, bevor er in die Zustände „True“ oder „False“ übergeht.

Continue Dieses Event nimmt eine Sonderstellung unter den Incoming Events ein, da es keinerlei Aktion auslöst, sondern ausschließlich dazu dient, eine Blockierung aufzuheben.

Tabelle 3.1 gibt einen Überblick über die definierten Events, ihre Quelle und die Möglichkeit, die Engine zu blockieren.

3.2.3.8 Architektur des Event-Frameworks

Zur einfachen Nutzung des Event-Modells ist es sinnvoll, ein Framework zu entwickeln, das die Anbindung verschiedener Komponenten erlaubt. Aus diesem Grund wird in [Khalaf u. a. (2007)] auch ein solches Framework vorgestellt. Es sieht die Nutzung einer Messaging-Infrastruktur vor, um die Events als Nachrichten zu verteilen. Das Framework übergibt die Events an anwendungsspezifische Komponenten zur weiteren Behandlung. Abbildung 3.7 zeigt die Architektur des Event Frameworks für BPEL.

Tabelle 3.1: Liste der Events

Event	Blockierend?	Quelle
Process_Deployed	-	Engine
Process_Instantiated	-	Engine
Instance_Running	-	Engine
Instance_Suspended	-	Engine
Instance_Terminated	-	Engine
Instance_Completed	-	Engine
Instance_Faulted	-	Engine
Activity_Ready	X	Engine
Activity_Executing	-	Engine
Activity_Executed	X	Engine
Activity_Complete	-	Engine
Activity_Dead_Path	-	Engine
Activity_Terminated	-	Engine
Activity_Faulted	X	Engine
Scope_Compensating	-	Engine
Scope_Compensated	-	Engine
Scope_Handling_Event	-	Engine
Scope_Event_Handling_Ended	-	Engine
Scope_Handling_Fault	X	Engine
Scope_Complete_With_Fault	X	Engine
Loop_Condition_True	X	Engine
Loop_Condition_False	X	Engine
Loop_Iteration_Complete	X	Engine
Link_Ready	-	Engine
Link_Evaluated	X	Engine
Compensate_Scope	-	Incoming Event
Fault_To_Scope	-	Incoming Event
Start_Activity	-	Incoming Event
Complete_Activity	-	Incoming Event
Continue_Loop_Execution	-	Incoming Event
Finish_Loop_Execution	-	Incoming Event
Continue_Loop	-	Incoming Event
Set_Link_State	-	Incoming Event
Continue	-	Incoming Event

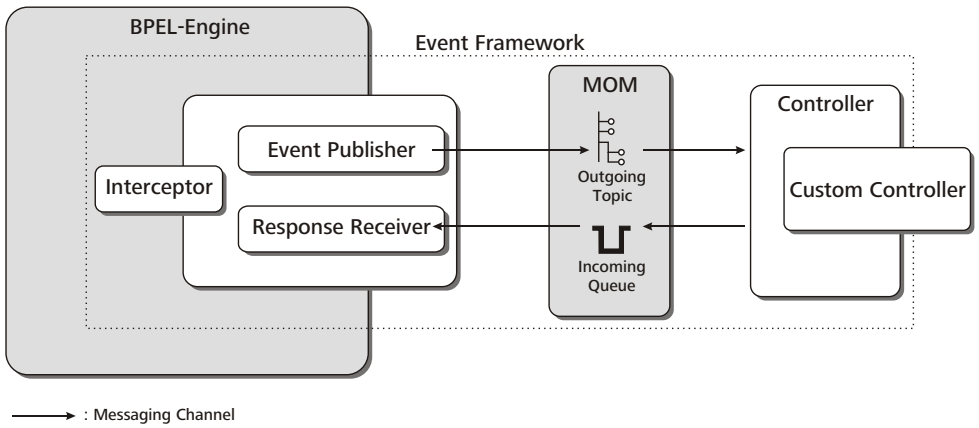


Abbildung 3.7: Architektur des Event Frameworks (Grau hinterlegte Komponenten sind nicht Bestandteil des Event Frameworks)

Der *Interceptor* fängt die Zustandsübergänge innerhalb der BPEL-Engine ab und erzeugt daraus Events. Der Event-Publisher kategorisiert die Events, versieht sie mit Kontextinformationen und publiziert sie auf einen Topic. Die Verteilung der Event-Nachrichten geschieht durch eine Message-Oriented-Middleware (MOM) an verschiedene, auf den Topic registrierte Empfänger. Den eigentlichen Kern des Frameworks bildet der *Controller*, der den Rahmen für die anwendungsspezifischen Komponenten zur Verarbeitung der Events (hier *Custom Controller* genannt) bildet. Der Controller entkoppelt die anwendungsspezifischen Implementierungen von der Kommunikation mit dem Messaging-System. Als Rückkanal für den Versand von Incoming Events aus dem Controller wird eine Queue genutzt, die vom *Response Receiver* abgehört wird. Der Response Receiver ist für die Verarbeitung der Incoming Events zuständig.

Für die Filterung der Events, die ein Custom Consumer empfangen möchte, wird der Selektor-Mechanismus der MOM genutzt. Hierzu werden die Events in verschiedene Klassen (wie z. B. Prozess-, Aktivitäts- und Link-Events) eingeteilt und mit entsprechenden Meta-Informationen gekennzeichnet.

Die Definition, welche Events blockierend sein sollen, erfolgt zur Deployment-Zeit des Prozesses durch einen zusätzlichen Deployment-Deskriptor, in dem die Elemente des Prozesses und die zugehörigen Events, die blockierend sein sollen, spezifiziert werden.

Die Tatsache, dass die Festlegung, welche Events blockierend sein sollen, beim Deployment des Prozesses erforderlich ist und somit Änderungen ein Redeployment des Prozesses erfordern, kann eine potentielle Einschränkung der Flexibilität des hier vorgestellten AOP-Ansatzes bedeuten, der gerade den Bedarf des Redeployments des Prozesses verhindern will. Aus diesem Grund wurden Mechanismen vorgesehen, die dieses Problem zu vermeiden helfen. Der Weaver bietet einen Mechanismus an, der durch Konfigurationsoptionen aktiviert

werden kann, mit dem der Weaver, wenn kein passender Aspekt für ein blockierendes Event gefunden wurde, durch ein spezielles Incoming Event (Continue) die Blockierung auflöst. Es ist also möglich, bereits beim Deployment eines Prozesses bestimmte Events, die man als Kandidaten für die Einwebung vermutet (hier wird die Obliviousness-Eigenschaft durchbrochen) als blockierend zu markieren und den Weaver entsprechend zu konfigurieren. Dies bedeutet zwar eine geringe Reduktion der Performanz, bietet jedoch einen wesentlichen Gewinn an Flexibilität.

3.2.3.9 Konsequenzen für die Architektur

Durch die Nutzung des Event-Frameworks stellt sich die Systemarchitektur wie in Abbildung 3.8 abgebildet dar.

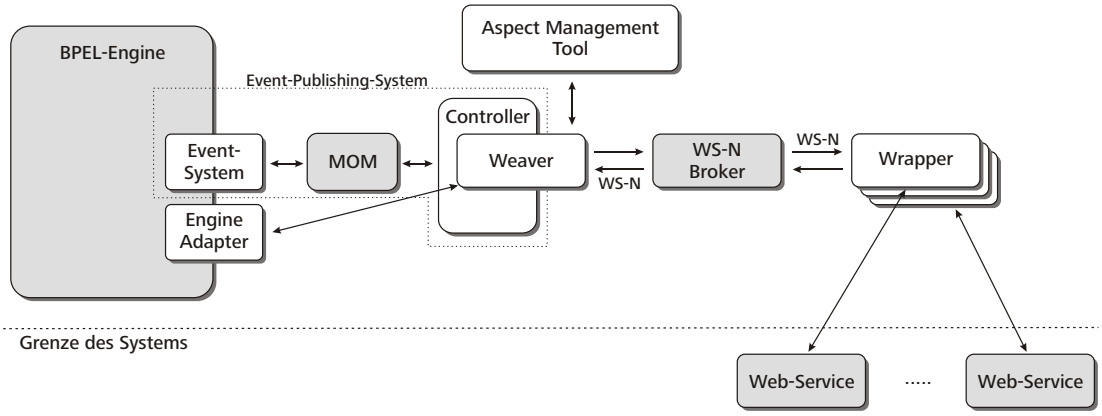


Abbildung 3.8: Systemarchitektur mit Event-Publishing-System (grau hinterlegte Komponenten sind nicht Bestandteil dieser Arbeit, die Abkürzung „WS-N" steht für WS-Notification, für Architektur des Event-Publishing-Systems siehe Abbildung 3.7)

Der *Weaver* ist eine Implementierung eines Custom-Controllers. Hierdurch wird eine Entkopplung des Weavers von der konkreten BPEL-Engine erreicht (wobei noch immer eine Abhängigkeit zum Engine-Adapter besteht, der enginespezifisch ist), wodurch er ohne große Änderungen an eine andere BPEL-Engine angepasst werden kann.

3.2.4 Konzeption des Weavers

Der Weaver bildet die Kern-Komponente der AOP-fähigen BPEL-Engine. Seine Aufgaben können der Beschreibung der konzeptionellen Architektur in Abschnitt 3.2.2 entnommen werden. Im Folgenden werden einige zentrale Konzepte des Weavers beschrieben. Zunächst wird auf das AOP-Modell näher eingegangen und die unterstützten Advice-Typen und

ihre Realisierung, das verwendete Joinpoint-Modell und die Spezifikation der Pointcuts beschrieben. Anschließend wird auf die Repräsentation und das Deployment der Aspekte eingegangen. Hieran schließt sich eine Betrachtung der Behandlung der Parameter der einzuwebenden Web-Service Operationen, der bei den Aufrufen möglicherweise auftretenden Ausnahmen und der Kompensation von eingewobenen Aspekten an. Anschließend werden die Kommunikationsschnittstellen des Weavers zum Wrapper und für das Deployment von Aspekten näher erläutert. Abschließend gibt es noch einen kurzen Abriss über ein Auditing-Konzept für eingewobene Aspekte.

3.2.4.1 Unterstützte Advice-Typen

Advices beschreiben die konkreten Arten, wie Erweiterungen in einen Prozess eingewoben werden können. Ein Advice ist in diesem Zusammenhang eine Operation eines Web-Service Ports, also eines konkreten Web-Service Endpunkts.

Eine Nutzung von Porttypen anstatt Ports wäre ebenfalls eine denkbare Alternative. Hierbei wäre es zum einen möglich, beim Deployment des Aspekts einen konkreten Port festzulegen, zum anderen könnte eine Service-Registry genutzt werden, um ausgehend vom Porttyp und evtl. spezifizierten weiteren Policies einen Port zu bestimmen. Diese Thematik ist jedoch nicht Kern dieser Arbeit, weshalb sie hier nicht weiter erörtert wird und der Weaver zunächst auf die Nutzung von Ports beschränkt wird.

Das hier verwendete Modell für die Advices orientiert sich an AspectJ (siehe Abschnitt 2.2.5). Folgende Typen von Advices werden vom Weaver unterstützt:

Before Der Advice wird vor der Ausführung der eigentlichen Aktivität ausgeführt.

After Der Advice wird nach der Ausführung der eigentlichen Aktivität ausgeführt.

Instead Der Advice ersetzt die Ausführung der eigentlichen Aktivität.

OnFault Der Advice wird ausgeführt, wenn die durch den Pointcut spezifizierte Aktivität durch einen Fault abgebrochen wurde. Die Ausführung findet hierbei vor der Ausführung des Fault-Handlers statt.

OnCompensation Der Advice wird ausgeführt, wenn der durch den Pointcut spezifizierte Scope kompensiert wird. Die Ausführung findet hierbei vor der Ausführung der Kompensation statt.

OnLinkEvaluation Der Advice wird ausgeführt, nachdem die Transition-Condition eines Links ausgewertet wurde. Hierbei kann der Wert der Transition-Condition durch das Resultat des Advice ersetzt werden.

Die Realisierung eines Instead-Advice erfolgt durch Nutzung des Complete_Activity Inco-

ming Events, um die Ausführung der zugehörigen Aktivität zu überspringen, indem sie in den Complete-Zustand überführt wird, ohne dass die Aktivität ausgeführt wird.

3.2.4.2 Joinpoint-Modell

Da das Weaving durch die Reaktion auf Events realisiert wird, wird das Joinpoint-Modell durch die verfügbaren Events bestimmt. Generell ist es möglich jedes, der in Abschnitt 3.2.3.1 beschriebenen, von der Engine generierten, Events als Joinpoint zu verwenden. Für einige der dort dargestellten Events ist der Nutzen im Kontext der AOP jedoch eher zweifelhaft, weshalb eine Beschränkung auf die wichtigsten Events stattfindet. Der Weaver kann bei Bedarf jedoch um weitere Events als Joinpoints ergänzt werden.

Viele der Events sind nur in der Verwendung mit einem speziellen Advice-Typ sinnvoll, weshalb man vereinfachend nur die Quellen der Events als nach außen sichtbares Joinpoint-Modell ansehen kann. Wohlgemerkt verwendet der Weaver intern die einzelnen Eventtypen als Joinpoints, nach außen hin werden die Events jedoch durch die Kombination aus Quelle (Aktivität oder Link) und Advice-Typ definiert, was zum einen Flexibilität bietet (einfacheres Umstellen des Advice-Typs, ohne dass Änderungen an den Pointcut-Ausdrücken erforderlich werden) und zum anderen die Analogie zum AspectJ Modell aufrechterhält. Tabelle 3.2 zeigt die einzelnen Joinpoints, Advice-Typen und zugehörigen Events.

Tabelle 3.2: Abbildung der Joinpoint- und Advice-Typen auf Events

Joinpoint	Advice-Typ	zugeh. Event
Aktivität	Before	Activity_Ready
Aktivität	After	Activity_Executed
Aktivität	Instead	Activity_Ready
Aktivität	OnFault	Scope_Handling_Fault (Scopes) bzw. Activity_Faulted (andere Aktivitäten)
Aktivität	OnCompensation	Scope_Compensating
Link	OnLinkEvaluation	Link_Evaluated

3.2.4.3 Spezifikation von Pointcuts

Die Spezifikation von Pointcuts wird für Aktivitäten und Links unterschiedlich gehandhabt. Links werden über ihren Namen im BPEL-Dokument identifiziert. Zur Spezifikation der Pointcuts für Aktivitäten kommt XPath (vgl. [Clark und DeRose (1999)]) zum Einsatz. Es ist hierfür besonders gut geeignet, da es zum Adressieren von Bestandteilen eines XML-Dokuments entworfen wurde. Der Weaver unterstützt für die Selektion von Aktivitäten den vollen Sprachumfang von XPath, so dass auch Wildcards, Shortcuts etc. verwendet werden

können. Hierdurch wird auch ein Quantifikationsmechanismus geboten, mittels dem mehrere Aktivitäten durch einen Pointcut-Ausdruck (siehe Listing 3.1) selektiert werden können. Bei der Spezifikation der Pointcuts für Links ist es ebenfalls möglich Wildcards zu verwenden.

3.2.4.4 Repräsentation der Aspekte

Um die Anforderung der Gruppierbarkeit von Aspekten zu erfüllen, werden Aspekte zwingend in sog. *Aspektgruppen* organisiert, die ähnlich einem Package in Java einen Namensraum definieren, in dem alle enthaltenen Aspekte liegen. Hierdurch wird zum einen erreicht, dass Namenskonflikte zwischen Aspekten vermieden werden können. Zum anderen wird durch sie auch ein Mechanismus zum untrennbaren Verbinden von Aspekten für das Deployment realisiert. Hierdurch wird es ermöglicht, die gewünschte Alles-oder-Nichts Semantik für zusammengehörige Aspekte zu realisieren. Hier endet die Analogie zu Java-Packages, da diese keine solchen semantischen Garantien anbieten.

Für die externe Repräsentation der Aspekte und Aspektgruppen wird WS-Policy (siehe Abschnitt 2.4.1) genutzt. Dies ist die Umsetzung der in Abschnitt 3.1.1 und Abbildung 3.1 auf Seite 36 beschriebenen Struktur eines Aspekts. Für die Repräsentation eines Aspekts werden spezielle Policy-Assertions definiert. Die zugehörige Schema-Definition der Assertions findet sich in Anhang A.

Listing 3.1: Struktur einer Aspekt-Definition

```
<wsp:Policy xmlns:wsu="http://docs.oasis-open.org/wss/
            2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
            wsu:Id="AspectGroupName">
 <wsp:All>
  <a4b:Aspect Id="..."?>*
   <a4b:Advice name="..."?>
    <a4b:When type="before | instead | after | ..."/>
    <wsa:EndpointReference>
     ...
     <a4b:WSDLLocation>...</a4b:WSDLLocation>
    </wsa:EndpointReference>
    <a4b:Operation name="..."
                   SuppressFault="false | true"?
                   SuppressResult="false | true"?/>
    <a4b:InputTransformation>...</a4b:InputTransformation>?
    <a4b:OutputTransformation>...</a4b:OutputTransformation>?
   </a4b:Advice>
   <a4b:Pointcut>
    <a4b:ProcessArtifact
         type="activity | link"
         identifier="..."/>
   </a4b:Pointcut>
```

```
23    </a4b:Aspect>
24   </wsp:All>
25  </wsp:Policy>
```

Die Aspektgruppe wird als <wsp:All> Policy-Operator (siehe Zeile 4 in Listing 3.1) abgebildet, der die in der Gruppe enthaltenen Aspekte (bzw. die zugehörigen Policy-Assertions) umschließt. Der Name der Aspektgruppe wird als Name der Policy über den WS-SecurityUtility Identifikationsmechanismus für Policies als Name der gesamten Policy (siehe Zeile 3) verwendet.

Die einzelnen Policies werden über die komplexen <a4b:Aspect> Policy Assertions (siehe Zeile 5) dargestellt. Hierbei wird der Name und der Namespace des Aspekts als qualifizierter Name (QName) über das Attribut Id dargestellt.

Eingeschachtelt in die <a4b:Aspect> Policy Assertion werden das Element für die Advices (Zeile 6) und die Elemente für die Pointcuts (Zeile 18) dargestellt.

Die Definition der Advices enthält die Art des Advices im Element <a4b:When>. Der einzuwebende Web-Services wird über eine im <a4b:Advice> Element eingebettete WS-Addressing Endpunktreferenz (vgl. [Box u. a. (2004a)]) spezifiziert. Um die Adresse des WSDL-Dokuments, das den Web-Service Endpunkt beschreibt, die standardmäßig nicht Teil einer WS-Addressing Endpunktreferenz ist, zu spezifizieren, wird das Element <a4b:WSDLLocation> in das Extensibility-Element der Endpunktreferenz eingefügt (Zeile 10). Das WSDL-Dokument ist notwendig, um Informationen über die für den Aufruf zu verwendenden Nachrichtenformate zu erhalten. Die Operation des Web-Service Ports, die aufgerufen werden soll, wird im Element <a4b:Operation> (Zeile 12) spezifiziert. Hier besteht ebenfalls die Möglichkeit die Unterdrückung des Rückgabewerts (Attribut SuppressResult in Zeile 14) der Operation zu definieren (näheres siehe Abschnitt 3.2.4.7) sowie die Unterdrückung der Propagation von Fehlern (Attribut SuppressFault in Zeile 13) innerhalb des Web-Service zurück in den Prozess (siehe Abschnitt 3.2.4.9) zu spezifizieren. Optionale Transformationen der eingehenden und ausgehenden Nachrichten können in den Elementen <a4b:InputTransformation> und <a4b:OutputTransformation> (Zeilen 15 und 16) spezifiziert werden (Details werden in Abschnitt 3.2.4.7 beschrieben).

Die Pointcuts werden durch das Element <a4b:Pointcut> spezifiziert. Die eigentliche Spezifikation der Pointcuts erfolgt im <a4b:ProcessArtifiact> Element. Der Typ des Pointcuts wird im Attribut type festgelegt, die Spezifikation des Pointcut-Ausdrucks wird im Attribut identifier vorgenommen.

3.2.4.5 Policy-Attachment Mechanismus

Die Verbindung einer Aspektgruppe mit einem oder mehreren Prozessen oder Prozess-Instanzen erfolgt durch einen Policy-Attachment Mechanismus (siehe Abschnitt 2.4.2). Da es für die Realisierung von dynamischem Weaving nicht praktikabel ist, das BPEL-Dokument zu modifizieren, um die Aspektbeschreibungen einzubetten, scheidet die Nutzung des XML Element Attachment Mechanismus aus. Aus diesem Grund kommt ein externer Attachment Mechanismus zum Einsatz. Hierbei werden beim Deployment eines Aspektes separate XML-Dokumente für eine Aspektgruppe erzeugt, die eine PolicyAttachment Beschreibung enthalten. Hierdurch wird die für dynamisches Weaving notwendige Flexibilität beim Deployment und Undeployment von Aspekten erreicht.

Die Definition der Policies für die Aspekte erfolgt im selben Dokument wie die Definition des PolicyAttachments. Die Spezifikation der Prozesse und Instanzen erfolgt über eine sog. *Domain Expression*. Diese wird speziell für die Nutzung als Attachment-Mechanismus für Aspekte an BPEL-Prozesse definiert. Listing 3.2 zeigt die Struktur eines solchen Attachments.

Listing 3.2: Struktur eines Policy-Attachments für Aspekte

```
1 <wsp:PolicyAttachment>
2  <wsp:AppliesTo>
3   <a4b:Processes>
4    (<a4b:Model name="..."/>*
5      | <a4b:Instance ID="..."/>*)
6   </a4b:Processes>
7  </wsp:AppliesTo>
8  <wsp:Policy>...</wsp:Policy>
9 </wsp:PolicyAttachment>
```

Die Domain-Expression wird durch das Element `<a4b:Processes>` (Zeile 3 in Listing 3.2) eingeleitet. In diesem kann eingeschachtelt entweder ein oder mehrere Exemplare des Elements `<a4b:Model>` (Zeile 4) enthalten sein, das ein ganzes Prozessmodell (bzw. genauer alle bestehenden und zukünftigen Instanzen, auf die der Aspekt anwendbar ist, siehe Abschnitt 3.2.4.6) referenziert. Alternativ können ein oder mehrere Exemplare des Elements `<a4b:Instance>` eingeschachtelt sein, das individuelle Prozessinstanzen, die nicht zwingend dem selben Prozessmodell zugehörig sein müssen, referenziert (auch hier gilt die in Abschnitt 3.2.4.6 beschriebene Bedingung). Prozessmodelle werden über ihren qualifizierten Namen (QName) referenziert. Die Identifikation von Prozessinstanzen erfolgt über ihre innerhalb der Engine eindeutige Prozessinstanz-ID (PIID).

Wenn man den Attachment-Mechanismus genau betrachtet, stellt man fest, dass er in gewisser Weise eine Erweiterung des in Abschnitt 3.2.4.3 beschriebenen Pointcut-Mechanismus ist und die Domain-Expression eigentlich ein Bestandteil der Pointcut-Spezifikationen sein

müsste. Die Trennung ist jedoch sinnvoll, um eine bessere Wiederverwendbarkeit der Aspekte zu erreichen und zu ermöglichen, Aspekte ohne Kenntnis der konkreten Prozessnamen und Instanz-IDs zu definieren und diese erst beim Deployment des Aspekts zu spezifizieren, was mehr Flexibilität bietet.

Eine bessere Methode zur Identifikation der Instanzen, in die ein Aspekt eingewebt werden soll, wäre es, eine Instanz über Variableninhalte, wie z. B. den Wert eines Correlation-Sets, zu identifizieren. Hierdurch ist es nicht mehr notwendig a priori die Prozessinstanz-ID kennen zu müssen. Eine solche Identifikation ist jedoch komplex, sowohl was die Spezifikation der Werte, als auch was den Abgleich zu Laufzeit angeht, der potentiell wenig performant ist.

Listing 3.3 zeigt ein Beispiel für eine Aspektgruppe in einem PolicyAttachment.

Listing 3.3: Beispiel eines Aspekts inkl. Attachment

```
<wsp:PolicyAttachment
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:AppliesTo>
    <a4b:Processes>
      <a4b:Model xmlns:ns1="http://mydotcom.com/services/order"
                 name="ns1:Demoprocess"/>
    </a4b:Processes>
  </wsp:AppliesTo>
  <wsp:Policy xmlns:wsu="http://docs.oasis-open.org/wss/
              2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
              wsu:Id="DemoAspects">
    <wsp:All>
      <a4b:Aspect xmlns:a4b="http://www.iaas.uni-stuttgart.de/a4b"
                  xmlns:ns1="http://mydotcom.com/services/order"
                  Id="ns1:ChristmasDiscount">
        <a4b:Advice Name="DiscountAdvice">
          <a4b:When type="After"/>
          <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org
                                /ws/2004/08/addressing">
            <wsa:Address>
              http://localhost:8081/DemoServices/services/discountPort
            </wsa:Address>
            <wsa:PortType>ns1:discountPT</wsa:PortType>
            <wsa:ServiceName wsa:PortName="discountPort">
              ns1:discountService
            </wsa:ServiceName>
            <a4b:WSDLLocation>
              http://localhost:8081/DemoServices/discountPort?wsdl
            </a4b:WSDLLocation>
          </wsa:EndpointReference>
          <a4b:Operation name="calculateDiscountedPrice"
                         SuppressFault="false"
                         SuppressResult="false"/>
        </a4b:Advice>
```

```
35          <a4b:Pointcut>
36            <a4b:ProcessArtifact type="activity" identifier=
37              "//invoke[@name='InvokeCalculationService']"/>
38          </a4b:Pointcut>
39        </a4b:Aspect>
40      </wsp:All>
41    </wsp:Policy>
42  </wsp:PolicyAttachment>
```

3.2.4.6 Deployment von Aspekten

Das Deployment der Aspekte in den Weaver erfolgt über eine Web-Service Schnittstelle sowie alternativ über Dateien, die beim Start des Weavers in einem Deployment-Verzeichnis gesucht werden. Der Web-Service bietet Möglichkeiten zum Deployment, Undeployment und Auflisten der deployten Aspektgruppen und wird detailliert in Abschnitt 3.2.4.13 beschrieben.

Beim Deployment wird geprüft, ob der durch den Aspekt referenzierte Prozess oder die spezifizierte Prozessinstanz bereits bekannt ist. Ist dies nicht der Fall, wird das Deployment durch den Weaver zurückgestellt, bis Informationen über den Prozess oder die Instanz verfügbar sind. Dieser Mechanismus wird *Defered Deployment* genannt.

Für Aspektgruppen ist eine Anforderung die „Alles-oder-Nichts“ Eigenschaft. Damit ist gemeint, dass eine Aspektgruppe nur dann deployt wird, wenn sichergestellt ist, dass alle enthaltenen Aspekte noch ausgeführt werden können. Damit ist ausgeschlossen, dass ein Aspekt sich auf eine Aktivität auswirkt, die in einer Instanz schon vollständig ausgeführt wurde und der Aspekt so für diese Instanz niemals ausgeführt werden kann. Dies kann wiederum zu Problemen führen, wenn die Advices anderer Aspekte der Gruppe davon abhängen, dass der Advice des nicht ausführbaren Aspekts zuvor ausgeführt wurde.

Um diese Eigenschaft sicherzustellen, löst der Weaver beim eigentlichen Deployment die gesamten Pointcuts in konkrete Elemente des Prozessmodells auf und prüft, ob es für jeden der Aspekte mindestens einen Pointcut gibt, der mindestens ein Element des Prozessmodells referenziert, das noch nicht in einem Endzustand (Completed, Faulted, Compensated etc.) ist. Dies wird bei Aspektgruppen, die für ganze Prozessmodelle deployt werden, für jede referenzierte Instanz, die noch nicht abgeschlossen ist und schon existiert, geprüft. Beim Deployment für ganze Prozessmodelle wird für alle bereits existierenden, noch nicht abgeschlossenen Instanzen einzeln geprüft, ob der Aspekt für sie anwendbar ist. Wenn dies nicht der Fall ist, wird die entsprechende Instanz für die Anwendung der Aspektgruppe ausgeschlossen.

Die Überprüfung der Anwendbarkeit eines Aspekts basiert auf der Analyse des Prozessmodells einer Instanz, bei der die Zustände der einzelnen Aktivitäten und Links verwendet werden. Es wird ein vereinfachter Kontrollfluss-Graph (hier werden keine Handler berücksichtigt) des Modells aufgebaut und die Erreichbarkeit einzelner Aktivitäten von noch nicht in Endzuständen befindlichen Aktivitäten aus untersucht. Für eine genauere Prüfung müsste der vollständige Kontrollflussgraph (wie z. B. bei [Ouyang u. a. (2005)] beschrieben) aufgebaut werden, was aber aus Gründen des Aufwands hier nicht betrachtet wird.

Diese Überprüfung gestaltet sich noch komplexer, wenn, wie in Abschnitt 3.2.4.5 skizziert, eine Identifikation des Prozesses anhand von Variableninhalten durchgeführt wird. Dies ist jedoch auch nicht Thema dieser Arbeit und wird deshalb hier nicht weiter betrachtet.

Die deployten Aspekte werden persistent im Dateisystem als PolicyAttachments abgelegt, um sicherzustellen, dass sie nach einem Neustart des Weavers noch vorhanden sind. Hierzu werden beim Neustart die deployten Aspekte wieder eingelesen. Hierbei ist zu beachten, dass beim Redeployment während des Neustarts erneut die Anwendbarkeit des Aspekts geprüft wird. Dabei besteht die Möglichkeit, dass eine Prozessinstanz, auf die die Aspektgruppe vor dem Neustart angewendet wurde, nun ausgeschlossen wird, da ein oder mehrere Aspekte nicht mehr als ausführbar eingestuft werden, da sie bereits während der vorhergehenden Ausführung ausgeführt wurden. Dieses Problem ließe sich nur durch intensive Datenhaltung über die Ausführung von Aspekten auf Instanzen lösen, beispielsweise durch Nutzung von Auditing-Daten des Weavers. Dieses Problem ist nicht Kern dieser Arbeit und wird aus diesem Grund nicht weiter verfolgt und bedarf weitergehenderer Betrachtung.

Beim Undeployment von Aspektgruppen werden diese aus dem Dateisystem entfernt und im Weaver deregistriert. Evtl. noch laufende Advices werden hierbei nicht abgebrochen, sondern werden zu Ende ausgeführt.

3.2.4.7 Handhabung der Parameter für Web-Service Aufrufe

Die Aufrufe der einzuwebenden Web-Services benötigen i. d. R. Parameterwerte. Diese müssen aus dem BPEL-Prozess, bzw. genauer gesagt aus der Instanz, für die die Einwebung erfolgt, entnommen werden. Hierfür sind einige Mechanismen vorstellbar. Zum einen besteht die Möglichkeit, die Namen von Variablen innerhalb des Prozesses bei der Definition des Advice anzugeben, was maximale Flexibilität bietet, jedoch problematisch sein kann, wenn ein Advice für verschiedene Prozessmodelle mit potentiell unterschiedlichen Variablennamen verwendet werden soll. Ein weiterer Ansatz ist es, grundsätzlich die Eingabe- und Ausgabe-Parameter der Aktivität, die die Quelle eines Events ist, zu übergeben und in Abhängigkeit vom Typ des Advice einen von beiden zu verwenden.

Dasselbe gilt für die Rückgabewerte der eingewobenen Web-Services. Auch hier kann

entweder eine bestimmte Variable spezifiziert werden, oder in Abhängigkeit vom Typ des Advice auf die Eingabe- oder Ausgabe-Variablen zurückgegriffen werden.

Der Weaver greift aus Gründen der Flexibilität und Wiederverwendbarkeit der Advices sowohl für Aufrufparameter als auch für die Rückgabewerte auf die zweite Strategie zurück. Hierzu wurden die in Tabelle 3.3 dargestellten Regeln für die Bestimmung der Parameterwerte definiert.

Tabelle 3.3: Regeln für Bestimmung der Parameterwerte

Advice-Typ	Quelle Eingabeparameter	Ziel Rückgabewert
Before	Input-Variable	Input-Variable
After	Output-Variable	Output-Variable
Instead	Input-Variable	Output-Variable
OnFault	Fault-Message	-
OnCompensation	-	-
OnLinkEvaluation	-	Wert der Transition-Condition

Der Zugriff auf die Variablen eines Prozesses allein ist nicht ausreichend, da hierdurch die Auswahl der einwebbaren Web-Services durch die Nachrichtenformate der Variablen sehr eingeschränkt ist. Aus diesem Grund ist es sinnvoll einen Mechanismus zum Außerkraftsetzen des Standardvorgehens für den Variablenzugriff vorzusehen. Zum einen ist es wünschenswert eine Möglichkeit zum expliziten Zugriff auf bestimmte Variablen innerhalb einer Instanz vorzusehen. Zum anderen ist ein Mechanismus zur Transformation der Nachrichtenformate wünschenswert. Hierfür kann beispielsweise XSLT als standardisierter Transformationsmechanismus für XML-Daten sinnvoll, da die Variablen innerhalb des Prozesses XML-Fragmente sind. Ein weiterer möglicher Ansatz wäre die Möglichkeit allgemeine BPEL-Assign Aktivitäten zu spezifizieren, um aus mehreren Variablen eine neue Nachricht aufzubauen. Hierfür existieren bereits Verfahren, weshalb hier nicht näher auf diesen Ansatz eingegangen wird.

Da die Transformation von Nachrichten eine Standardfunktionalität von Web-Service Middleware wie z. B. einem *Enterprise Service Bus* (vgl. [Chappell (2004)]) ist, wird im Weaver nur die Möglichkeit zum Zugriff auf definierbare Variablen vorgesehen. Die Definition der Transformation der Eingabe- und Ausgabeparameter innerhalb des Advice ist jedoch durch ein Erweiterbarkeitselement so gestaltet, dass später weitere Transformationsmechanismen nachgerüstet werden können. Um festzustellen, ob eine Transformation des Nachrichtentyps notwendig ist, wird überprüft, ob die Nachrichtentypen der aus dem Prozess enthaltenen Variablen und der Parameter der Web-Service Operation übereinstimmen. Zu dieser Überprüfung wird die Typinformation aus der BPEL-Engine und die WSDL-Beschreibung des aufzurufenden Web-Service Endpunkts verwendet.

Da es nicht immer wünschenswert und sinnvoll ist, das Resultat eines Web-Service Aufrufs in den Prozess zurückzupropagieren, enthält der Weaver eine Möglichkeit, die Übergabe

des Resultats gemäß der oben definierten Regeln zu unterdrücken.

3.2.4.8 Präzedenz von Aspekten

Für die Ausführung von Aspekten bedarf es Regeln, die die Reihenfolge festlegen, in der mehrere Aspekte ausgeführt werden, die an derselben Stelle eingewoben werden. Dies ist insbesondere für den korrekten Entwurf von Aspekten wichtig, da hierfür die Abhängigkeiten zwischen Aspekten von Bedeutung sind. An diesem Punkt wird die Analogie des AOP-Mechanismus zu AspectJ gebrochen, da in AspectJ die Präzedenzregeln teilweise über die Position der Advices innerhalb einer Quelldatei definiert werden, was im hier verwendeten Ansatz nicht anwendbar ist.

Für die Präzedenz von verschiedenen Advices ist es sinnvoll, das Joinpoint-Modell genauer zu betrachten. Die einzigen Advice-Typen, die zueinander in Konflikt stehen, sind Before- und Instead-Advices, alle anderen Advice-Typen sind auf eigene Events abgebildet. Aus diesem Grund wurden die folgenden zwei Präzedenzregeln für Aspekte definiert.

- Before-Advices werden immer vor Instead-Advices ausgeführt.
- Zwischen Advices desselben Typs wird keine feste Reihenfolge definiert, so dass die Ausführungsreihenfolge implementierungsspezifisch ist.

Die Übergabe von Parametern und Rückgabewerten an die einzuwebenden Web-Services ist auch vom Präzedenz-Mechanismus betroffen. Aus Gründen der Komplexität wird die Übergabe so realisiert, dass nicht jedes Resultat eines Web-Service in den Prozess zurückpropagiert wird, sondern das des letzten eingewobenen Advice, der nicht den Rückgabewert unterdrückt.

Um mehr Flexibilität bei der Behandlung von Advices zu erlangen wäre es sinnvoll einen Prioritätsmechanismus zu realisieren, über den explizit die Präzedenz von Advices des selben Typs definiert werden kann. Ebenso wäre es wünschenswert, einen Mechanismus zu realisieren, über den eine Übergabe von Parameter- und Rückgabewerten zwischen Advices möglich wird. Diese Funktionalitäten sind jedoch jenseits des in dieser Arbeit betrachteten Themas und bedürfen weitergehenderer Betrachtung.

3.2.4.9 Behandlung von Ausnahmen

Da während der Ausführung der eingewobenen Web-Services Ausnahmen auftreten können, ist es notwendig, einen Mechanismus zur Propagation der Faults in die Prozessinstanz, in der die Einwebung stattfand, zu definieren. Ein nahe liegender und trivialer Ansatz

hierzu ist, auf eine solche Propagation komplett zu verzichten, was jedoch evtl. zur Folge hat, dass spätere Advices unzutreffenderweise ausgeführt werden. Ein anderer Ansatz ist es, die Faults zurück in die Engine zu propagieren und als Fault des Prozesselements, an dem die Einwebung stattfand in die Engine einzubringen. Dies hat den Vorteil, dass eine korrekte Fehlerbehandlung stattfinden kann. Allerdings hat dies den Nachteil, dass evtl. im Prozess Fault-Handling Logik vorgesehen werden muss, um die Faults in den eingewobenen Web-Services zu behandeln. Dies ist insbesondere deshalb problematisch, da bei der Erstellung des Prozesses potentiell kein Wissen über die Web-Services, die zur Laufzeit dynamisch eingewoben werden, vorhanden ist. Dies widerspricht der Obliviousness-Eigenschaft (Abschnitt 2.2.3), die fordert, dass ein Komponentenprogramm (in diesem Fall der BPEL-Prozess) kein Wissen über die eingewobenen Aspekte besitzen muss.

Der im Weaver gewählte Ansatz ist, generell die oben beschriebene Propagation von Ausnahmen in den Prozess vorzunehmen. Um die Obliviousness-Eigenschaft zu erhalten, besteht darüber hinaus die Möglichkeit, für die einzuwebenden Operation zu definieren, dass in ihr auftretende Ausnahmen unterdrückt und nicht propagiert werden sollen.

3.2.4.10 Kompensationsmechanismus

Da Workflows und auch WS-Flows teilweise langlaufende Interaktionen abbilden, ist ein Transaktionsmodell wie z. B. atomare Transaktionen (ACID Transaktionen) nicht geeignet. Aus diesem Grund werden für langlaufende Workflows *kompensationsbasierte Transaktionsmodelle* eingesetzt (vgl. [Leymann und Roller (2000)]). Hierbei werden Aktionen nicht durch ein Rollback wie in atomaren Transaktionen zurückgenommen, sondern durch eine sog. *Kompensationsoperation* aufgelöst. Ein Beispiel für eine Kompensationsoperation wäre die Rücknahme einer Hotelbuchung durch eine Stornierung.

Auch in BPEL kommt ein kompensationsbasiertes Transaktionskonzept zum Einsatz. Aus diesem Grund ist es erforderlich, eine Integration der eingewobenen Aspekte in den Kompensationsmechanismus zu schaffen. Ein nahe liegender Ansatz wäre es, für jeden Aspekt einen Kompensations-Handler zu definieren. Wenn eine Kompensation innerhalb der Prozessinstanz stattfindet, werden die Kompensations-Handler der für die Instanz eingewobenen Advices ausgeführt. Da BPEL jedoch Scopes als Grenzen für die Behandlung von Kompensationen bietet, ist auch eine Unterstützung für Scopes bei diesem Vorgang durch den Weaver notwendig. Darüber hinaus ist die Frage, wie mehrfache Anwendungen eines Advice innerhalb einer Prozessinstanz gehandhabt werden, sowohl was die Reihenfolge der Kompensations-Handler angeht, als auch was die Problematik angeht, dass Advices durch den Quantifizierungsmechanismus potentiell in verschiedenen Scopes angewendet werden können. Hierdurch wird dieser Mechanismus relativ komplex und es wird erforderlich, dass der Weaver in größerem Umfang instanzbasierte Zustandsinformationen für die einzelnen deployten Aspekte mitführen muss. Im Weaver wird deshalb ein anderer Ansatz verfolgt.

Der vom Weaver realisierte Ansatz gestaltet sich wie folgt: Die Kompensation eines Scopes wird als eigener Joinpoint über die OnCompensation-Advices verfügbar gemacht. Die von den Advices ausgeführten Web-Services sind für die Kompensation der innerhalb des Scopes angewendeten Aspekte verantwortlich. Komplexere Kompensationsaktionen lassen sich beispielsweise durch das Einweben eines anderen BPEL-Prozesses realisieren, der die notwendigen Operationen durchführt. Dieser Ansatz ist leichtgewichtiger als der oben skizzierte, da der Weaver so keinen Zustand für die Aspekte halten muss und keine Probleme mit der Ausführungsreihenfolge der Kompensations-Handler der einzelnen Advices auftreten können.

Da die Kompensation von großer Bedeutung für die korrekte Ausführung von Workflows ist, bedarf es vertiefender Betrachtungen, die den Rahmen dieser Arbeit jedoch überschreiten.

3.2.4.11 Zugriff auf Informationen der Engine

Die Anbindung des Weavers an die Engine erfolgt über den Engine-Adapter (siehe Abbildung 3.8 auf Seite 51). Über diese Schnittstelle erhält der Weaver Informationen über die deployten Prozesse, die zugehörigen Prozessmodelle, die Zustände der einzelnen Instanzen usw. Diese Informationen sind für den Betrieb des Weavers notwendig.

Die Prozessmodelle werden als BPEL-Dokumente übertragen. Die Informationen über den Zustand von Instanzen werden durch um Zustandsinformationen angereicherte BPEL-Dokumente abgebildet. Diese Informationen werden nur beim Deployment und der Instanziierung eines Prozesses sowie beim initialen Anbinden des Weavers an den Engine-Adapter übertragen. Zur Aktualisierung der Zustandsinformationen werden die Events des Event-Systems genutzt.

3.2.4.12 Kommunikation des Weavers mit dem Wrapper

Die Kommunikation zwischen Weaver und Wrapper erfolgt über WS-Notification. Es wird für jeden Pointcut ein eigener Topic erzeugt, auf den der Weaver wenn ein Advice für diesen Pointcut ausgeführt werden soll (d. h. ein entsprechendes Event wurde empfangen), eine Nachricht publiziert, die den Wrapper dazu veranlasst, einen Aufruf auszuführen. Auf diese Weise werden nur für Events, für die ein Advice existiert, Nachrichten an den Wrapper erzeugt. Für jeden Topic ist nur ein Wrapper registriert (was sinnvoll ist, da doppelte Ausführungen der einzuwebenden Web-Services nicht wünschenswert sind), so dass eine Partitionierung der existierenden Pointcut-Topics unter den verschiedenen Wrappern entsteht. Als Rückkanal für die Antworten der Wrapper wird wiederum ein WS-Notification Topic verwendet, für den der Weaver als Consumer subskribiert ist und

auf dem alle existierenden Wrapper ihre Antwortnachrichten publizieren. Die generelle Struktur der Kommunikation ist nochmals in Abbildung 3.9 dargestellt.

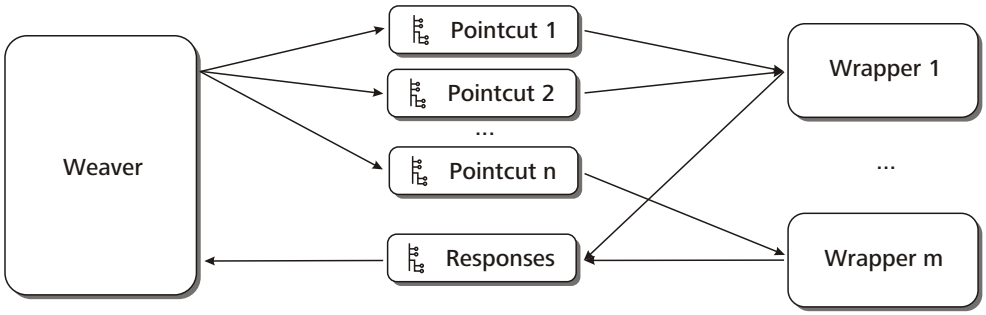


Abbildung 3.9: Struktur der Kommunikation zwischen Weaver und Wrapper (der WS-Notification Broker wurde hier zur besseren Übersicht weggelassen)

Da WS-Notification als asynchrone one-way Kommunikationsform keine Zuordnung der Antwortnachrichten zu den versendeten Nachrichten bietet, wird ein expliziter Korrelationsmechanismus eingesetzt. In die vom Weaver verschickte Nachricht wird eine vom Weaver generierte zufällige Kennung, als sog. *Correlation-ID* (vgl. [Hohpe und Woolf (2003)]) aufgenommen. Diese sendet der Wrapper mit der Antwortnachricht an den Weaver zurück. Hierdurch kann der Weaver die Antwort zuordnen und korrekt weiterverarbeiten.

Das vom Weaver für die Kommunikation mit dem Wrapper verwendete Nachrichtenformat ist in Listing 3.4 dargestellt.

Listing 3.4: Struktur einer Invocation Notification Message

```
<a4bn:InvocationNotification
      xmlns:a4bn="http://www.iaas.uni-stuttgart.de/projects/
                  2006/a4b-notification"
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <a4bn:CorrelationData>...</a4bn:CorrelationData>
  <a4bn:MessageData messageType="...">
        ...
  </a4bn:MessageData>
  <wsa:EndpointReference>
    ...
    <a4bn:WSDLLocation>...</a4bn:WSDLLocation>
  </wsa:EndpointReference>
  <a4bn:Operation name="..."/>
</a4bn:InvocationNotification>
```

Die Korrelationsinformationen werden im Element `<a4bn:CorrelationData>` angegeben. Das Element `<a4bn:MessageData>` enthält die Parts, die als Parameter für den Aufruf der Operation des Web-Service genutzt werden sollen. Der Typ der Nachricht wird im Attribut

messageType als qualifizierter Name (QName) angegeben. Der aufzurufende Web-Service wird über eine WS-Addressing Endpunktreferenz (vgl. [Box u. a. (2004a)]) spezifiziert. Im Erweiterungselement der Endpunktreferenz wird das Element <a4bn:WSDLLocation> eingeschachtelt, das die Adresse der WSDL-Beschreibung des Web-Service Endpunkts enthält. Die aufzurufende Operation des Web-Service wird im Element <a4bn:Operation> angegeben.

Der Aufbau der Antwortnachrichten des Wrappers an den Weaver ist in Listing 3.5 dargestellt.

Listing 3.5: Struktur einer Response Notification Nachricht

```
<a4bn:ResponseNotification
      xmlns:a4bn="http://www.iaas.uni-stuttgart.de/projects/
                  2006/a4b-notification"
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <a4bn:CorrelationData>...</a4bn:CorrelationData>
  <a4bn:MessageData messageType="...">
    ...
  </a4bn:MessageData>
  <a4bn:FaultData>?
    <a4bn:FaultName>...</a4bn:FaultName>
    <a4bn:FaultInfo>...</a4bn:FaultInfo>
    <a4bn:FaultDetail>...</a4bn:FaultDetail>
    <a4bn:FaultMessageData>
        ...
    </a4bn:FaultMessageData>
  </a4bn:FaultData>
</a4bn:ResponseNotification>
```

Das Element <a4bn:CorrelationData> enthält die Korrelationsdaten der Invocation Notification Nachricht, auf die diese Response Notification Nachricht sich bezieht. Im Element <a4bn:MessageData> sind die Parts enthalten, die das Resultat des Aufrufs des Web-Service sind. Optional kann eine Response Notification Nachricht zusätzlich das Element <a4bn:FaultData> enthalten, das Informationen über einen im aufgerufenen Web-Service aufgetretenen Fehler enthält. Das Element <a4bn:FaultName> enthält den qualifizierten Namen des Typs des aufgetretenen Fehlers, <a4bn:FaultInfo> enthält eine kurze Beschreibung des aufgetretenen Fehlers, die im Element <a4bn:FaultDetail> ausführlicher wiedergegeben ist. Die vom Wrapper empfangene Fault-Message wird im Element <a4bn:FaultMessageData> eingefügt.

In Anhang B ist die genaue Schema-Definition für die Notification Nachrichten wiedergegeben.

Die Kommunikation des Weavers mit den Wrapper erfolgt über einen blocking receive

Mechanismus, d. h. der Weaver wartet solange, bis eine Antwortnachricht des Wrappers eintrifft. Es wird ein Timeout-Mechanismus genutzt, um eine unendliche Blockierung des Weavers zu verhindern, wenn der Wrapper nicht verfügbar ist. Nach Ablauf des Timeout verfährt der Weaver so, als ob eine Nachricht eingegangen ist, aber sowohl Resultat als auch etwaige Fehler unterdrückt werden. Dies bedeutet, dass im Falle eines Timeouts keinerlei Propagation zurück in die BPEL-Engine stattfindet.

3.2.4.13 Deployment-Schnittstelle des Weavers

Der Weaver bietet eine Web-Service Schnittstelle, für das Deployment der Aspekte an. Sie stellt die in Tabelle 3.4 angegebenen Operationen bereit.

Tabelle 3.4: Beschreibung des Deployment Web-Service

Operation	Beschreibung
`deployAspectGroup`	Deployt die übergebene Aspektgruppe. Als Parameter wird ein DOM-Dokument erwartet, das ein PolicyAttachment-Dokument, das die Aspektgruppe und Referenzen auf die Prozesse bzw. Instanzen, für die sie deployt werden soll, enthält. Als Rückgabewert wird ein Boolean-Wert zurückgegeben, der den Erfolg des Deployment widerspiegelt. Wenn ein Defered Deployment notwendig werden sollte, wird hier trotzdem ein erfolgreiches Deployment gemeldet.
`undeployAspectGroup`	Undeployt eine Aspektgruppe, die über ihren qualifizierten Namen spezifiziert wird. Als Rückgabewert wird ein Boolean-Wert geliefert, der angibt, ob das Undeployment erfolgreich war.
`listDeployedAspectGroups`	Gibt eine Liste der qualifizierten Namen aller deployten Aspektgruppen zurück.

Die Deployment-Schnittstelle wird vom Aspekt Management Tool (siehe Abschnitt 3.2.6) genutzt.

3.2.4.14 Auditing-Mechanismen für den Weaver

Auditing von Workflows ist ein wichtiger Baustein für die Optimierung von Workflows durch Business-Process Reengineering. Darüber hinaus hat Auditing auch Bedeutung für die Erfüllung gesetzlicher Auflagen, beispielsweise des Sarbanes-Oxley Act. Unter Auditing von Workflows versteht man hierbei das Aufzeichnen aller relevanten Ereignisse im Leben einer Workflow-Instanz (vgl. [Leymann und Roller (2000)]). Hierdurch wird nachvollziehbar, was in einem Workflow wann und warum geschehen ist.

Vor diesem Hintergrund ist es notwendig, dass auch die durch AOP hervorgerufenen Aktionen im Lebenszyklus einer Workflow-Instanz aufgezeichnet werden.

Der Weaver selbst enthält keine Vorkehrungen um Audit-Informationen zu speichern. Es ist jedoch möglich Auditing-Daten zu sammeln, indem man die Nachrichten, die über die Topics zwischen Weaver und Wrapper ausgetauscht werden, auswertet. Um dies zu ermöglichen, wurde der Rückkanal vom Wrapper zum Weaver ebenfalls als Topic realisiert. die Aufzeichnung von Auditing-Informationen kann in diesem Szenario erfolgen, indem auf allen Topics eine Auditing-Komponente subskribiert wird, die die ausgetauschten Nachrichten aufzeichnet.

Die momentan in den ausgetauschten Messages enthaltenen Informationen sind allerdings noch nicht ausreichend, um sinnvolles Auditing zu erlauben. Für ein sinnvolles Auditing müssten zusätzlich Zeitstempel, Informationen über die Prozessinstanz und Aktivität, die für einen Web-Service Aufruf verantwortlich waren, etc. aufgezeichnet werden (vgl. [Leymann und Roller (2000)]). Darüber hinaus kann es auch sinnvoll sein aufzuzeichnen, wann welcher Aspekt deployt und undeployt wurde, und welcher Benutzer dies veranlasst hat.

Die Integration der Auditinformationen für die Einwebung von Aspekten in den Audit-Trail der BPEL-Engine ist nicht trivial, da sie außerhalb der Engine anfallen. Hierdurch wird ein eigener Audit-Trail für den Weaver oder ein Mechanismus in der Engine zur Erfassung der externen Audit-Informationen notwendig. In beiden Fällen bedarf es eines Mechanismus zur Korrelation der Auditdaten der beiden Quellen.

Die genaue Behandlung des Themenbereichs Auditing ist noch zu klären, da es nicht Thema dieser Arbeit ist.

3.2.5 Konzeption des WS-Notification Wrappers

Der Wrapper dient dazu, den Weaver von den eigentlichen Web-Service Aufrufen zu entkoppeln. Er fungiert als *Gateway* (vgl. [Fowler (2002)]) zwischen dem Weaver und den einzuwebenden Web-Services und behandelt die Details der Web-Service Aufrufe für den Weaver.

Die vom Weaver publizierten Invocation Request Nachrichten werden vom Wrapper empfangen und verarbeitet. Er führt einen dynamischen Aufruf des einzuwebenden Web-Service durch. Hierzu wird die WSDL-Beschreibung des einzuwebenden Ports analysiert und ein entsprechender Aufruf ausgeführt. Der Wrapper unterstützt sowohl Document/Literal als auch RPC/Encoded Endpunkte. Als Binding wird ausschließlich SOAP über HTTP unterstützt, es ist jedoch möglich, andere Bindings später hinzuzufügen. Dies kann entweder

in der Art erfolgen, dass ein Wrapper mehrere Bindings unterstützt und das zu nutzende aus dem WSDL-Dokument bestimmt. Ein anderer Ansatz ist es mehrere Wrapper, die verschiedene Bindings unterstützen anzubieten, die auf die entsprechenden Pointcut-Topics subskribiert sind.

Die für die Kommunikation zwischen Wrapper und Weaver verwendeten Nachrichtenformate werden in Abschnitt 3.2.4.12 beschrieben.

3.2.6 Konzeption des Aspekt Management Tools

Zur Verwaltung und Definition von Aspekten wird ein graphisches Werkzeug eingesetzt, da die manuelle Erstellung der Aspekte in Form von Policy-Dokumenten sehr aufwändig ist. Die durch das Werkzeug unterstützten Use-Cases sind in Abbildung 3.10 dargestellt.

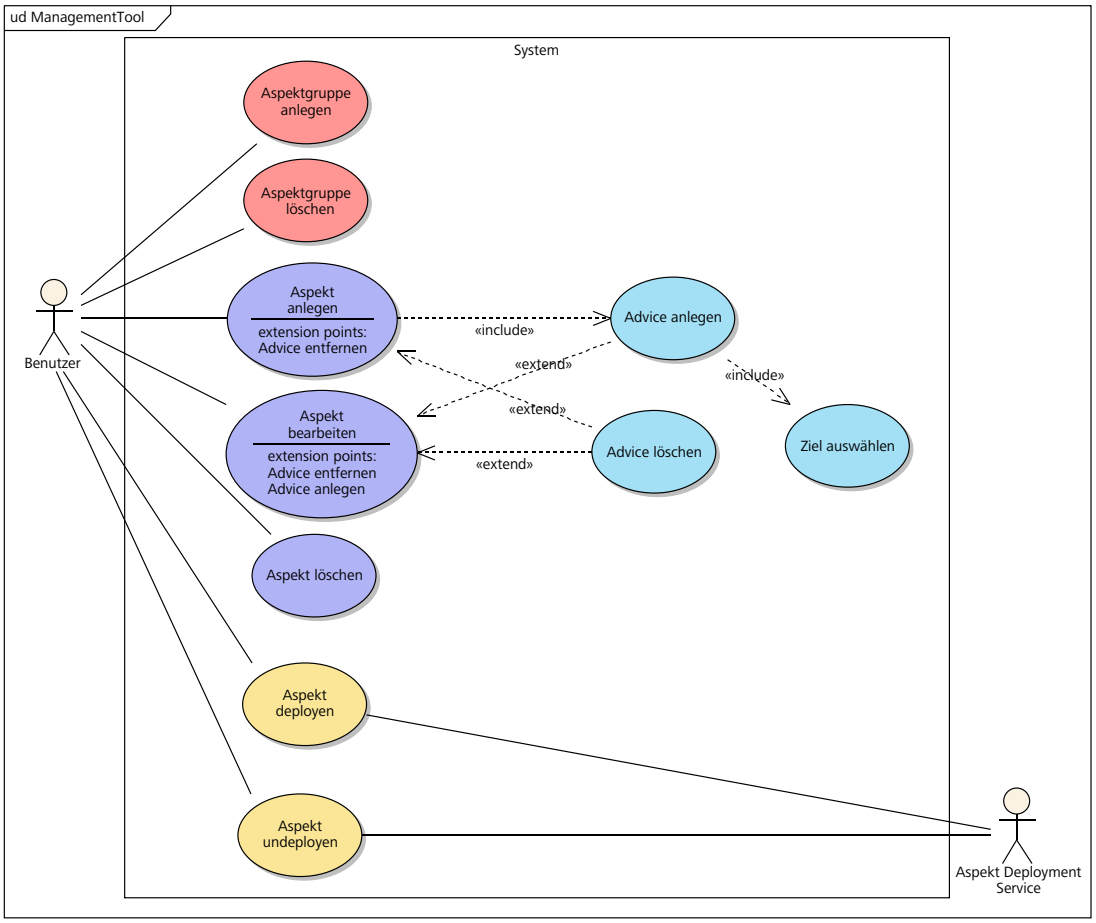


Abbildung 3.10: Use-Case Diagramm für das Aspect Management Tool

3.2.6.1 Benutzerschnittstelle

Die Interaktion mit dem Aspekt Management Tool erfolgt über eine grafische Benutzerschnittstelle (GUI) in Form eines Fat-Client. Abbildung 3.11 – 3.16 zeigen Screenshots der Anwendung. Alle in Abbildung 3.10 dargestellten Use-Cases werden durch die grafische Benutzeroberfläche unterstützt.

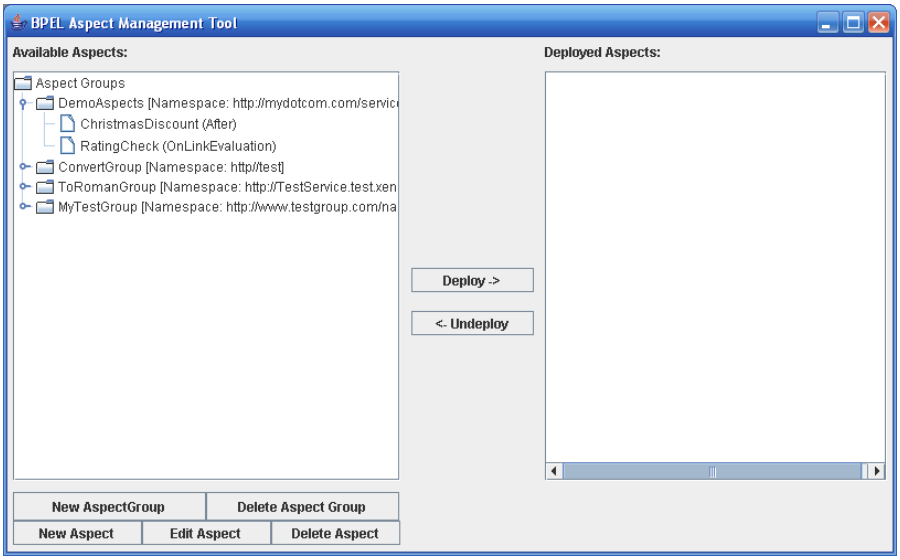


Abbildung 3.11: Hauptfenster des Aspekt Management Tools

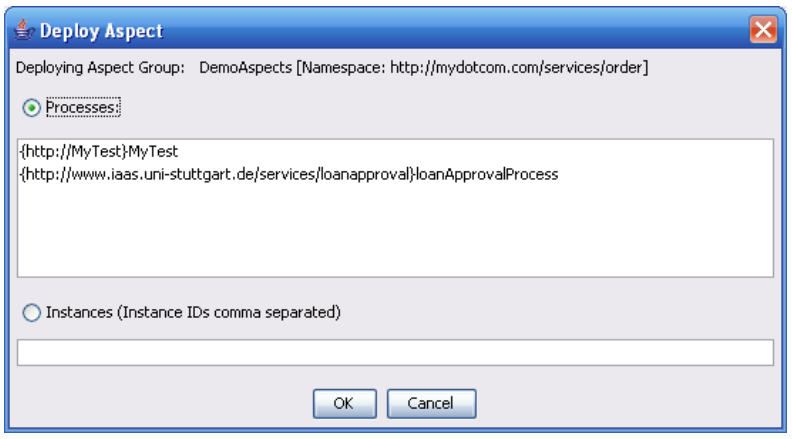


Abbildung 3.12: Fenster für das Deployment einer Aspektgruppe

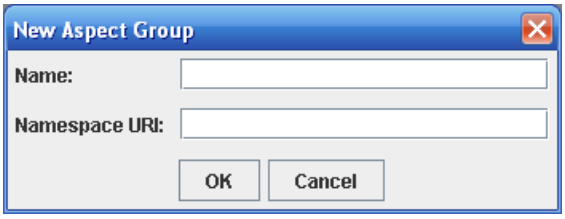


Abbildung 3.13: Fenster zum Anlegen einer neuen Aspektgruppe

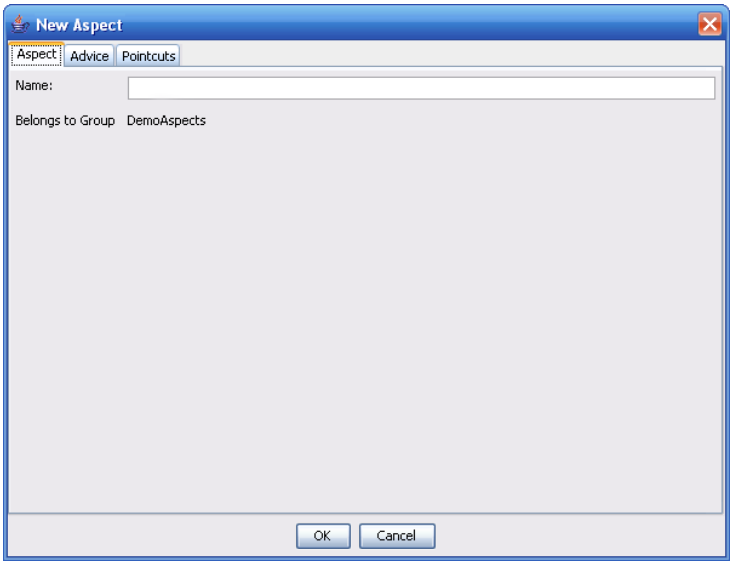


Abbildung 3.14: Fenster zum Anlegen eines Aspekts

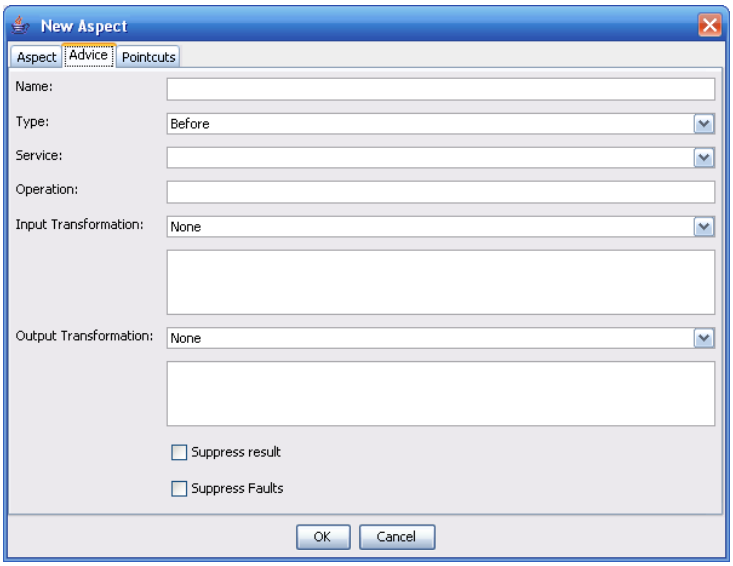


Abbildung 3.15: Fenster zum Definieren eines Advice

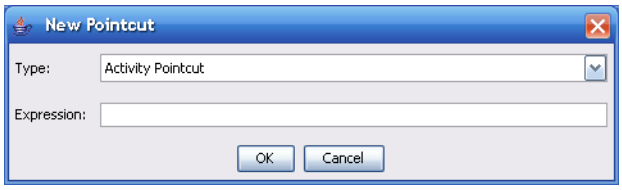


Abbildung 3.16: Fenster zum Anlegen eines Pointcut

3.2.6.2 Quelle der verfügbaren Web-Service Endpunkte

Das Aspekt Management Tool benötigt Informationen über die zur Verfügung stehenden Web-Service Endpunkte, die als Advice eingewebt werden können. Die verfügbaren Endpunkte werden im Dateisystem abgelegt und beim Start des Werkzeugs eingelesen. Sie liegen in Form von Dateien mit der Endung `.epr` vor, die jeweils eine WS-Addressing Endpunktreferenz enthalten. Diese Endpunktreferenz enthält zusätzlich einen Verweis auf die Adresse der WSDL-Beschreibung des referenzierten Service, die im Element `<a4b:WSDLLocation>` angegeben wird.

Listing 3.6: Beispiel für eine vom Aspekt Management Tool genutzte Endpunktreferenz

```
<wsa:EndpointReference
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <wsa:Address>
    http://localhost:8081/DemoServices/services/creditworthinessPort
  </wsa:Address>
  <wsa:PortType xmlns:ns1="http://mydotcom.com/services/order">
    ns1:creditworthinessPT
  </wsa:PortType>
  <wsa:ServiceName xmlns:ns1="http://mydotcom.com/services/order"
                   wsa:PortName="creditworthinessPort">
    ns1:creditworthinessService
  </wsa:ServiceName>
  <a4b:WSDLLocation xmlns:a4b="http://www.iaas.uni-stuttgart.de/a4b">
    http://localhost:8081/DemoServices/services/creditworthiness.wsdl
  </a4b:WSDLLocation>
</wsa:EndpointReference>
```

Es ist auch denkbar, eine UDDI-Registry als Quelle für die Web-Service Endpunkte zu nutzen. Dieser Ansatz wird jedoch aus Gründen der Komplexität hier nicht verwendet. Ein anderer, weniger komplexer Ansatz wäre beispielsweise die Anwendung eines Find&Bind Verfahrens, wie es von [Wutke (2006)] und [Karastoyanova u. a. (2005a)] beschrieben wird.

3.2.6.3 Speicherung der erstellten Aspekte

Die im Aspekt Management Tool verwalteten Aspekte werden persistent im Dateisystem abgelegt. Die Speicherung erfolgt als serialisierte WS-Policy Dokumente. Jede Aspektgruppe wird in eine eigene Datei serialisiert, deren Name den qualifizierten Namen der Aspektgruppe widerspiegelt. Beim Löschen einer Aspektgruppe wird auch die zugehörige Datei aus dem Dateisystem entfernt. Ob ein Aspekt deployt ist oder nicht hat dabei keinen Einfluss auf die Löschung, da die deployten Aspekte durch den Weaver verwaltet werden, es ist also möglich einen Aspekt aus dem Aspekt Management Tool zu löschen und ihn weiterhin im Weaver deployt zu lassen.

3.2.6.4 Zugriff auf Informationen über deployte Prozesse

Für das Deployment der Aspektgruppen in die BPEL-Engine sind Informationen über die in der BPEL-Engine deployten Prozessmodelle notwendig, damit entsprechende PolicyAttachments aufgebaut werden können. Die Erfassung der Informationen über zu den in der Engine deployten Prozessen durch den Benutzer ist fehlerträchtig und wenig komfortabel, weshalb ein automatischer Mechanismus verwendet wird.

Für den Zugriff auf die Informationen über die deployten Prozessmodelle wird eine weitere Schnittstelle zur BPEL-Engine verwendet, über die die deployten Prozessmodelle aufgelistet werden können. Das Aspekt Management Tool nutzt diese Liste und stellt sie dem Benutzer beim Deployment von Aspektgruppen zur Auswahl zur Verfügung (siehe Abbildung 3.12 auf Seite 69).

KAPITEL 4

Umsetzung

In diesem Kapitel wird zunächst auf die für die Umsetzung eingesetzten Technologien eingegangen. Im Anschluss wird ein Überblick über die Realisierung gegeben. Anschließend wird die Realisierung des BPEL Event Frameworks näher beschrieben. Daran schließt sich die detaillierte Beschreibung der Bestandteile des Systems an.

4.1 Eingesetzte Technologien

4.1.1 Java

Die Implementierung verwendet die Programmiersprache Java, da die bestehenden Komponenten bereits in Java erstellt wurden und eine sehr große Auswahl an frei verfügbaren Bibliotheken für Java existiert. Konkret wurde Java Version 5.0 (vgl. [Gosling u. a. (2005)]) eingesetzt, da die mit dieser Version eingeführten Spracherweiterungen mehr Typsicherheit und bessere Lesbarkeit bringen.

Für den Aspekt Weaver, der als Web-Applikation entworfen wurde, wird die Java 2 Enterprise Edition 1.4 (vgl. [Sun Microsystems (2003)]) genutzt.

4.1.2 BPEL-Engine

Für die Realisierung des AOP-Mechanismus ist es zunächst notwendig, eine BPEL-Engine auszuwählen, auf der er aufbauen soll. In Abschnitt 3.1.2 wurden die Anforderungen an eine solche Business-Process Engine beschrieben. Es bot sich an, die von [Wutke (2006)],

[Nitzsche (2006)] und [Wetzstein (2006)] erweiterte ActiveBPEL[1] 1.2 Engine (im Folgenden als *XEngine* bezeichnet) zu verwenden, da sie zum einen im Sourcecode vorliegt, durch die Arbeiten von Nitzsche und Wutke sehr gut dokumentiert ist und auch schon Kommunikationsmechanismen enthält, um Events innerhalb der Engine nach außen zu publizieren. Sie unterstützt Monitoring und realisiert die von [Karastoyanova u. a. (2006)] beschriebenen Konzepte. Auf dieser baut die Realisierung der AOP-fähigen BPEL-Engine auf.

ActiveBPEL ist eine von ActiveEndpoints[2] entwickelte und vertriebene BPEL-Engine. ActiveBPEL ist dual-lizensiert zum einen unter einer kommerziellen Lizenz und zum anderen unter der GNU General Public License (GNU GPL) verfügbar.

Parallel zu dieser Arbeit wurde eine neue Version der XEngine entwickelt, die auf ActiveBPEL 2.0 basiert und Auditing unterstützt (vgl. [Deshpande (2007)]), welche ebenfalls weitgehend mit dem hier entwickelten Aspect-Weaver zusammenarbeitet, jedoch kleine Einschränkungen besitzt, die in Abschnitt 4.5.10 und 5.2 beschrieben werden.

4.1.3 Spring Framework 1.2

Das Spring Framework[3] ist ein Java/J2EE Framework, das mit dem Ziel entwickelt wurde die Entwicklung von Enterprise-Applikationen zu erleichtern und die Entwickler vom Ballast der Nutzung von Enterprise Java Beans zu befreien. Es basiert auf dem von [Johnson (2002)] in seinem Buch skizzierten Framework. Das Herz von Spring bildet ein *Inversion of Control Container* (*IoC-Container*). IoC, auch bekannt als Dependency Inversion Prinzip (vgl. [Martin (2002)]), erreicht eine lose Kopplung von Softwarekomponenten, indem die Abhängigkeiten zwischen den Komponenten nicht mehr in den Komponenten selbst, sondern durch einen IoC-Container verwaltet werden.

Um die Abhängigkeiten den Komponenten bekannt zu machen verwendet das Spring Framework dabei die Dependency Injection Methode.

Hierbei injiziert der Container aktiv die Abhängigkeiten in die Komponenten, indem er sie ihnen bei der Instanziierung übergibt, oder später durch Setter-Methoden setzt. Dieser Mechanismus wird häufig von Frameworks genutzt und wird von [Fowler (2004)] näher beschrieben. Der Vorteil von *Dependency Injection* ist, dass die Komponenten keinerlei Abhängigkeiten zum genutzten IoC-Container haben müssen und so eine sehr gute Wartbarkeit erreicht werden kann.

Durch die Nutzung des Dependency Injection Ansatzes, müssen alle Komponenten gegen

[1] http://www.activebpel.org/
[2] http://www.active-endpoints.com/
[3] http://www.springframework.org/

definierte Schnittstellen implementiert werden, was zu einer geringen Kopplung zwischen den Komponenten führt. Aus diesem Grund kann man mit dem IoC-Ansatz erstellte Architekturen auch als eine Art lokaler SOA ansehen, wobei die Komponenten die Rolle der Service-Requester und der Container die Rolle der Service-Provider und Service-Registry übernehmen.

Das Komponentenmodell von Spring ist sehr leichtgewichtig, da es auf POJOs (*Plain old Java Objects*), also ganz normalen Java-Klassen basiert. Die Komponenten können also auch ohne das Spring-Framework wiederverwendet werden. Um die Möglichkeit der Dependency Injection sicherzustellen, stellt Spring bestimmte Benennungs-Konventionen auf, an die sich die Klassen halten müssen, damit die Abhängigkeiten injiziert werden können.

Über den IoC-Container hinaus bietet Spring noch eine Reihe von weiteren Funktionalitäten, die die Entwicklung von Anwendungen mit EJBs, JMS, O/R-Mappern, Web-Services, anderen Remoting-Implementierungen etc. vereinfachen, die für die Umsetzung des Lösungsansatzes jedoch größtenteils nicht verwendet wurden.

4.1.4 WS-Messenger

WS-Messenger[4] (vgl. [Huang u. a. (2006)]) ist eine unter einer BSD-artigen Lizenz verfügbare Implementierung eines Brokers für WS-BrokeredNotification und WS-Eventing, die am Extreme Lab der Indiana University entwickelt wird. Er erlaubt die Transformation der Nachrichtenformate zwischen WS-Eventing und WS-Notification, so dass Consumer beider Spezifikation auf dem gleichen Topic subskribiert sein können. Es ist auch möglich eine Message-Oriented-Middleware, wie NaradaBrokering, ActiveMQ oder openJMS als Grundlage von WS-Messenger zu verwenden. Auch ohne eine MOM zu verwenden bietet WS-Messenger persistente Nachrichten, wofür eine MySQL-Datenbank als Grundlage verwendet wird.

4.1.5 Apache Neethi

Apache Neethi[5], früher unter dem Namen „Apache WS-Commons Policy“ bekannt, ist eine Implementierung des WS-PolicyFramework, die unter der Apache License 2.0 frei verfügbar ist. Es bietet Funktionen zum Laden, Speichern, Modifizieren und Verarbeiten von Policies an.

In der momentan vorliegenden Version 1.0.1 bietet Apache Neethi keine Unterstützung für

[4] http://www.extreme.indiana.edu/xgws/messenger/
[5] http://ws.apache.org/commons/neethi/

Namespaces für in Policy-Assertions eingeschachtelte Elemente beim Parsen und Serialisieren von WS-Policies. Darüber hinaus wird auch nur ein Attachment-Mechanismus für die Einbettung von WS-Policies in WSDL-Dokumenten unterstützt.

Da die Unterstützung von Namespaces für den skizzierten Lösungsansatz bedeutsam ist, wurde entsprechender Unterstützungs-Code per AspectJ eingewoben. Die so gemachten Erweiterungen sind nicht fest mit Apache Neethi verbunden und konnten ohne Änderungen nach einem Upgrade von Version 1.0 auf Version 1.0.1 weiterverwendet werden.

Die Unterstützung für Policy-Attachment Dokumente wurde nicht in Apache Neethi nachgerüstet, sondern durch eine eigene Implementierung realisiert (siehe Abschnitt 4.3.2).

4.1.6 Apache Axis 1.4

Apache Axis[6] ist ein Web-Service Framework, das unter der Apache License 2.0 frei verfügbar ist. Es basiert auf den verbreiteten Web-Service Standards wie SOAP und WSDL. Axis bietet Unterstützung für die Entwicklung von Web-Services und Web-Service Clients, sowie eine Reihe von Hilfsfunktionen, um die Nutzung zu vereinfachen.

4.1.7 AspectJ

AspectJ[7] ist eine aspektorientierte Erweiterung von Java und hat sich mittlerweile zum de facto Standard für AOP in Java entwickelt. AspectJ geht auf die Entwicklung von Gregor Kiczales am Xerox PARC zurück. Eine ausführliche Beschreibung von AspectJ und seines Programmiermodells findet sich in Abschnitt 2.2.5 und bei [Gradecki und Lesiecki (2003)].

4.1.8 Java Message Service

Java Message Service (vgl. [Sun Microsystems (2002)]), kurz JMS, ist eine API für die Nutzung einer Message-Oriented-Middleware in Java Umgebungen und Bestandteil der Java Enterprise Edition (JEE). JMS unterstützt sowohl Queue- als auch Topic-basierte Kommunikation.

Als JMS-Provider wurde für die Realisierung ActiveMQ[8] genutzt.

[6] http://ws.apache.org/axis/
[7] http://www.eclipse.org/aspectj/
[8] http://www.activemq.org/

4.1.9 Apache Tomcat

Als Laufzeitumgebung für den Aspect Weaver und die BPEL-Engine wurde Apache Tomcat[9] verwendet. Tomcat ist ein Web-Container (vgl. [Sun Microsystems (2003)]), der unter der Apache License 2.0 frei verfügbar ist, und implementiert in der Version 5.x die Servlet 2.4 und JSP 2.0 Spezifikationen, die Bestandteil der Java 2 Platform Enterprise Edition (vgl. [Sun Microsystems (2003)]) sind.

4.1.10 DOM4J

DOM4J[10] ist eine sehr einfach zu benutzende XML-Bibliothek für Java. Es integriert sich wesentlich besser in Java als das W3C Document Object Model (vgl. [Le Hors u. a. (2004)]), da es konsequent Java Collections verwendet und viele Hilfsmethoden anbietet. Es bietet Unterstützung für XML, XPath und XSLT an und ist zudem performanter als andere, ähnlich mächtige XML Bibliotheken.

4.2 Überblick

Das AOP-System wird als verteilte Anwendung realisiert. In Abbildung 4.1 ist die Laufzeitarchitektur des Systems dargestellt.

Die Abbildung zeigt den Extremfall der Verteilung, es ist ebenso möglich, bestimmte Komponenten in eine gemeinsame Ausführungsumgebung zu platzieren.

Die XEngine wird als Web-Applikation innerhalb eines J2EE Web-Containers ausgeführt. Sie kommuniziert über einen JMS-Broker, in diesem Fall ActiveMQ, mit dem Aspekt Management Tool und dem Weaver. Der Weaver wird ebenfalls als Web-Applikation umgesetzt und innerhalb eines Web-Containers betrieben. Als WS-Notification Broker für die Kommunikation mit dem Wrapper kommt WS-Messenger zum Einsatz, der als stand-alone Applikation läuft, jedoch bei Bedarf ebenfalls in einem Web-Container betrieben werden kann. Das Aspekt Management Tool und der Wrapper werden als stand-alone Applikationen realisiert, wobei das Aspekt Management Tool eine Swing-basierte GUI nutzt.

[9] http://tomcat.apache.org/
[10] http://www.dom4j.org/

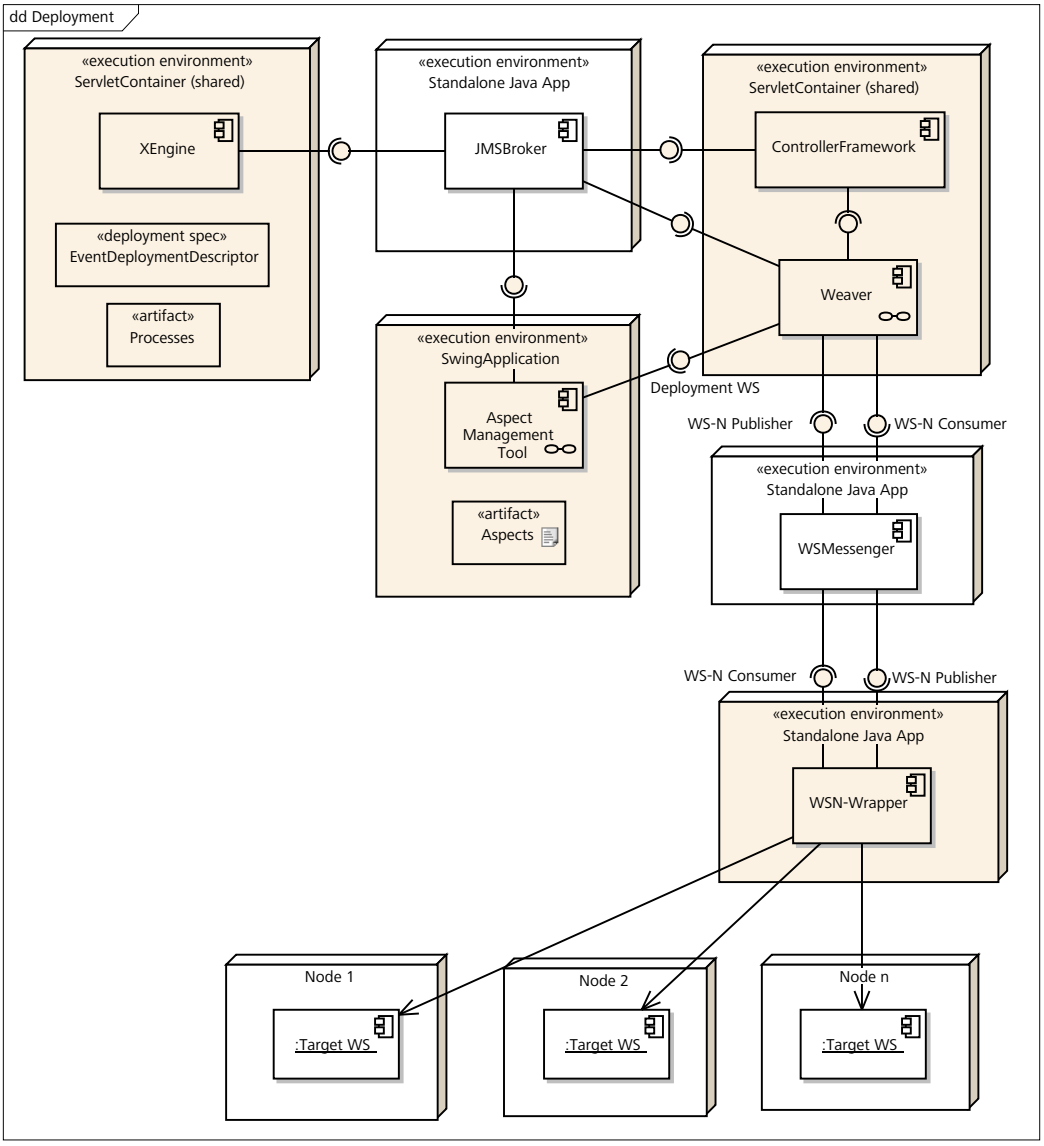


Abbildung 4.1: Laufzeitarchitektur des Systems

4.3 Datenmodell

4.3.1 Domänenmodell

Das von Weaver und Aspekt Management Tool verwendete Domänenmodell ist die Repräsentation des in Abschnitt 3.1.1 und 3.2.4 beschriebenen Modells für Aspekte. Abbildung 4.2 zeigt das verwendete Domänenmodell.

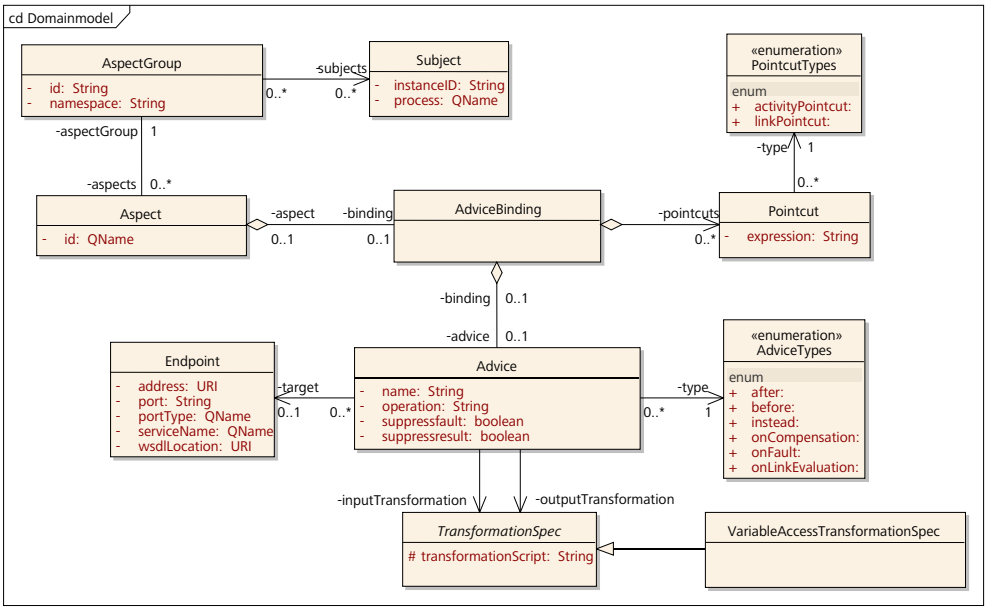


Abbildung 4.2: Domänenmodell für die AOP-fähige BPEL Engine

4.3.2 Serialisierung / Deserialisierung von Aspekten

Für die persistente Speicherung von Aspekten im Dateisystem und für die Übertragung von Informationen über den Deployment Web-Service bedarf es einer serialisierten Form der im Domänenmodell gespeicherten Informationen.

Der Serialisierungsmechanismus wird über das Interface `AspectSerializer` abgebildet, das Methoden für die Serialisierung und Deserialisierung von Aspektgruppen bereitstellt. Da Aspektgruppen, je nach dem, ob sie deployt sind oder nicht, sowohl als reine WS-Policy als auch als WS-PolicyAttachment serialisiert werden können, sind Implementierungen für Serializer für beide Formate notwendig. In Abbildung 4.3 ist dies dargestellt.

Für die Serialisierung nach WS-Policy wird Apache Neethi verwendet, da es ein einfach

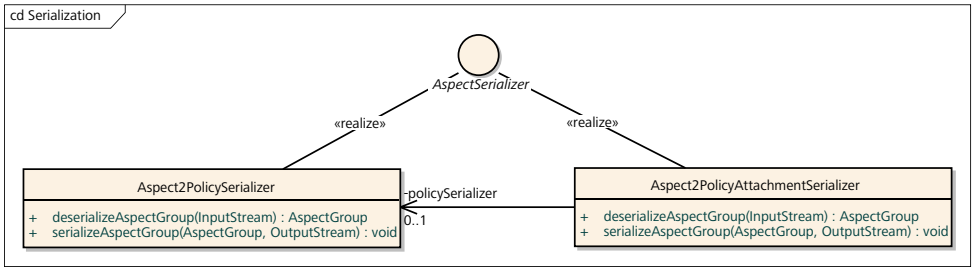


Abbildung 4.3: Klassendiagramm des Serialisierungsmechanismus

zu nutzendes Programmiermodell zur Beschreibung von WS-Policies anbietet. Leider bietet Apache Neethi bislang (Stand Version 1.0.1) keine Unterstützung für externe Policy Attachments. Für die Serialisierung in WS-PolicyAttachments fand sich auch keine andere geeignete Bibliothek, so dass nur eine Eigenimplementierung übrig blieb. Wie man Abbildung 4.3 entnehmen kann, nutzt diese die mittels Apache Neethi realisierte Serialisierung nach WS-Policy. Die von Apache Neethi erzeugte WS-Policy wird in ein DOM4J Dokument überführt, um die für das PolicyAttachment notwendigen Elemente ergänzt und anschließend serialisiert.

Die Deserialisierung erfolgt auf demselben Weg, wie die oben beschriebene Serialisierung.

Der Serialisierungsmechanismus ist auf Erweiterbarkeit ausgelegt, so dass ohne Probleme neue Serialisierungsformate hinzugefügt werden können. Um eine neue Implementierung hinzuzufügen, bedarf es nur einer Anpassung der Konfiguration des Spring-Frameworks.

4.4 BPEL Event Framework und Controller

In diesem Abschnitt wird die Realisierung des in Abschnitt 3.2.3.1 beschriebenen Event-Frameworks beschrieben. Zunächst wird auf die Integration in die bestehende XEngine eingegangen. Anschließend wird der Aufbau des Controller-Frameworks vorgestellt (siehe auch [Paluszek (2007)]).

4.4.1 Integration in die XEngine

Die Integration der für die Publikation der internen Events der Engine notwendigen Logik erfordert einige Änderungen an der Engine. Diese Logik ist ein typisches Beispiel für einen Crosscutting-Concern, so dass sich die Nutzung von AOP anbietet, um diese Erweiterungen

modular vornehmen zu können. Dieser Ansatz wurde bereits von [Houspanossian und Cilia (2005)] zur wartungsfreundlichen Erweiterung einer BPEL-Engine verwendet.

Die von [Wutke (2006)] und [Nitzsche (2006)] beschriebenen Erweiterungen der ActiveBPEL Engine werden soweit sinnvoll verwendet. Abbildung 4.4 zeigt, wie sich die Erweiterungen in die Architektur der XEngine einfügen.

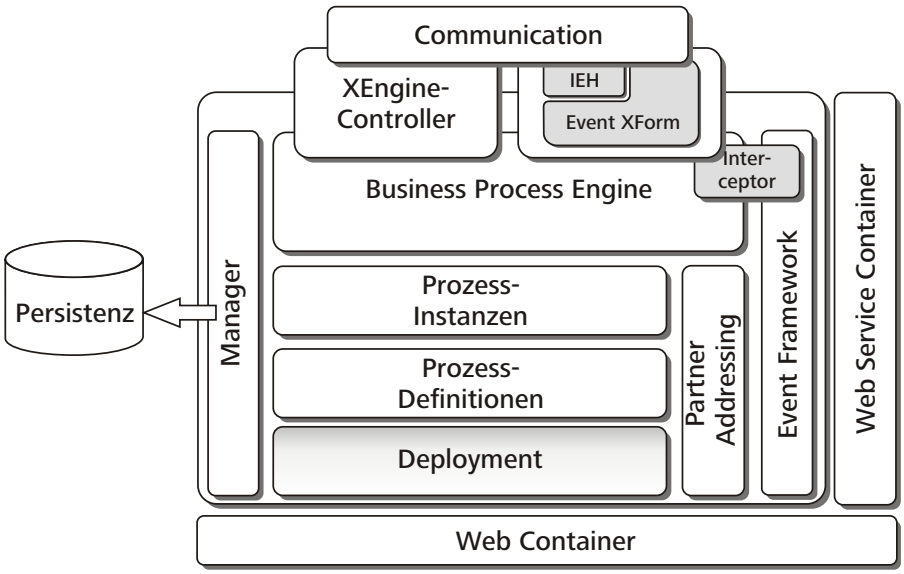


Abbildung 4.4: Integration des Event Frameworks in die XEngine, die grau hinterlegten Komponenten sind neu oder werden erweitert.

Die Komponente *Interceptor* dient zum Sammeln der Events. Zum einen werden die bereits von der Engine bereitgestellten Events (im Detail bei [Wutke (2006)] beschrieben) verwendet, soweit sie in das in Abschnitt 3.2.3.1 beschriebene Event-Modell passen. Hierfür wird vom Interceptor der in der Engine bereits vorhandene Listener-Mechanismus genutzt. Zum anderen wird an bestimmten Stellen innerhalb der eigentlichen Business Process Engine Code eingewoben, der entsprechende Events erzeugt. Die für den Interceptor notwendigen Änderungen wurden in mehrere AspectJ Aspekte modularisiert, die an den entsprechenden Stellen eingewoben werden.

Die Kernkomponente des Event Frameworks ist die *Event Transformation* Komponente. Sie ist für die Umwandlung der vom Interceptor gesammelten XEngine-spezifischen Events aus dem Event Framework der XEngine sowie der vom Interceptor generierten Events in portable BPEL Eventnachrichten zuständig. Sie baut Java-Objekte als Nachrichten für die Events auf, die zusätzlich einige Kontextinformationen zu einem Event enthalten. Dies sind zum einen Informationen über den Prozess und die Instanz, aus der ein Event stammt, aber auch Informationen zur Quelle eines Events, ebenso wie Variableninhalte

oder Fault-Informationen, im Fall von Events, die mit Faults zu tun haben. Die Klassen der aufgebauten Event-Objekte sind nach Typen in einer Vererbungshierarchie angeordnet, sie ist in Abbildung 4.5 dargestellt.

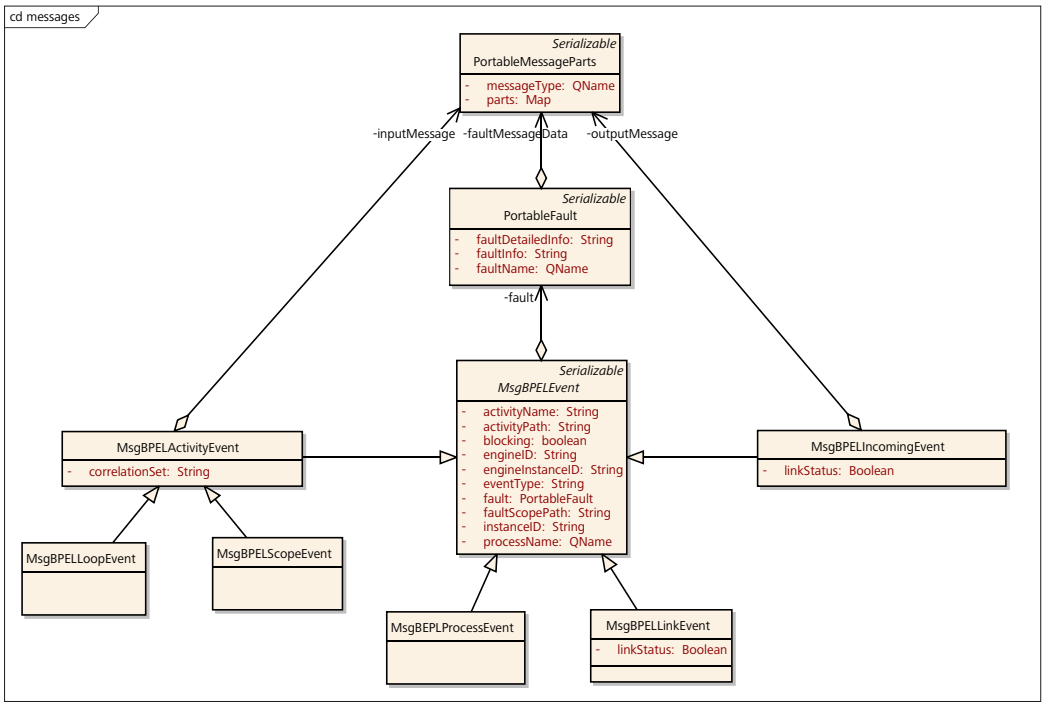


Abbildung 4.5: Hierarchie der Eventmessages

Zur Publikation der Event-Nachrichten wird die *Communication* Komponente der XEngine (für Details siehe [Wutke (2006)]) verwendet. Sie veröffentlicht die Nachrichten auf einen JMS-Topic, auf den die einzelnen Instanzen des *Controllers* (siehe Abbildung 4.6, nicht zu verwechseln mit dem XEngine Controller) subskribiert sind. Die versendeten Nachrichten werden über eine Message-Property entsprechend dem Typ des enthaltenen Objekts gekennzeichnet, um die Filterung der Nachrichten durch den JMS-Provider, entsprechend der von den Controllern spezifizierten Selektoren, zu ermöglichen. Zur Kommunikation werden die selben Kanäle wie bereits vom Monitoring-Werkzeug für die XEngine (vgl. [Nitzsche (2006)]) verwendet. Durch die Kennzeichnung der Nachrichten durch Nachrichtentypen kommt es hierbei zu keinen Konflikten. Abbildung 4.6 fasst die Kommunikationsbeziehungen nochmals zusammen.

Der *Incoming Event Handler* wird ebenfalls per AOP in die Communication Komponente der XEngine eingewoben und ergänzt den dortigen Message-Handler (vgl. [Wutke (2006)]) um die Behandlung der neuen Nachrichtentypen für die Incoming Events. Die Behandlung der einzelnen Events wird durch ein Strategy-Pattern (vgl. [Gamma u. a. (1995)]) realisiert, wobei für jeden Incoming Event Typ eine eigene Strategy genutzt wird. Hierdurch wird es

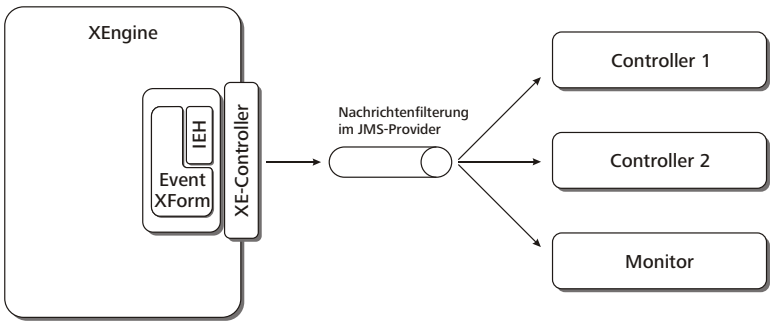


Abbildung 4.6: Kommunikation des Event Frameworks (XE-Controller steht für XEngine-Controller)

sehr einfach, neue Incoming Events hinzuzufügen.

Die *Deployment* Komponente der XEngine wird erweitert, um die Deployment-Deskriptoren zu lesen, die das Blockierungsverhalten für bestimmte Events und Eventquellen (siehe Abschnitt 3.2.3.1) beschreiben. Hierfür wird per AOP eine neue Komponente eingewoben, die nach dem Einlesen des ActiveBPEL Deployment-Deskriptors einen weiteren Deployment-Deskriptor einliest, der die oben genannten Informationen zu blockierenden Events enthält. Er ist in der Datei `META-INF/process.edd` innerhalb der `.bpr` Datei (Business Process Archive) des BPEL-Prozesses enthalten.

Listing 4.1: Beispiel für einen Event Deployment Descriptor

```
<edd:DeploymentData
  xmlns:edd="http://www.iaas.uni-stuttgart.de/
             XEngine/EventDeploymentData">
  <edd:EventDefinition>
    <edd:Node>
      /process/flow/receive[@name='receiveInput']
    </edd:Node>
    <edd:Event>Activity_Ready</edd:Event>
    <edd:Event>Activity_Executed</edd:Event>
    <edd:Blocking>true</edd:Blocking>
  </edd:EventDefinition>
</edd:DeploymentData>
```

4.4.2 Realisierung des Controller-Frameworks

Der Controller wird als Framework realisiert, das die anwendungsspezifischen Implementierungen (hier *Custom-Controller* genannt) ausführt. Hierbei gilt das für Frameworks typische „Hollywood-“ oder Inversion-of-Controll Prinzip, d. h. dass die anwendungsspezifische

Komponente ausschließlich durch das Framework aktiviert wird und nicht umgekehrt. Die Architektur des Frameworks ist in Abbildung 4.7 dargestellt.

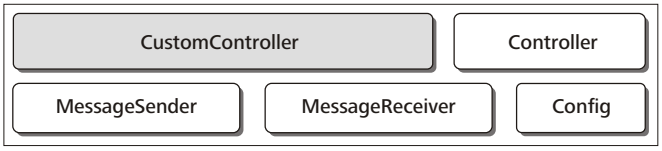


Abbildung 4.7: Schichtenarchitektur des Controller-Frameworks (grau hinterlegte Komponenten sind anwendungsspezifisch)

Der *Controller* ist für die Initialisierung des gesamten Frameworks zuständig und erzeugt bzw. bindet die anderen Komponenten.

Der *Message Receiver* wird vom JMS-Provider aufgerufen, wenn eine neue Nachricht, die den Kriterien des Selektors entspricht, auf dem JMS-Topic publiziert wurde. Er packt das in der Nachricht enthaltene Message-Objekt aus und übergibt es dem Custom-Controller, den er vom Controller zugewiesen bekam. Zusätzlich erstellt er ein opakes MessageContext-Objekt, das Korrelationsinformationen, Antwortadressen etc. enthalten kann und vom Message Sender für den korrekten Versand der Antworten des Custom-Controller verwendet wird. Hierdurch wird der Custom-Controller vom gesamten notwendigen Wissen über die genutzte MOM entkoppelt, so dass ein Wechsel von JMS zu anderen Technologien problemlos möglich ist, ohne Änderungen an bestehenden Custom-Controller Implementierungen zu erfordern.

In der Komponente *Config* wird die Konfiguration des Controller-Frameworks gekapselt. Über sie wird u. a. die Anbindung des Frameworks an den JMS-Provider definiert. Es wird eine Konfigurationsdatei im üblichen Java Properties-Format verwendet.

Der *Custom-Controller* ist die anwendungsspezifische Implementierung, die zur Verarbeitung der Events genutzt wird. Sie wird vom Message Receiver aktiviert, wenn eine neue gültige Nachricht empfangen wurde. Um Incoming Events an die Engine senden zu können, kann der Custom-Controller den Message Sender verwenden, der die MOM-spezifische Behandlung der Nachrichten vornimmt.

Die Realisierung des Controller-Frameworks erfolgt in Java. In Abbildung 4.8 ist das Design des Controller-Framework dargestellt.

Die Konfiguration, welcher Custom-Controller genutzt werden soll, geschieht über einen Plugin-Mechanismus oder alternativ über das Spring Framework, das die Klasse des Custom-Controller in die Controller Komponente injizieren kann. Voraussetzung für beide Mechanismen ist, dass jede einzubindende Komponente das Interface `CustomController` implementieren muss. Die zentrale Methode für die Verarbeitung der Events durch die Implementierungen des Custom-Controllers ist `handleEventMessage`. In ihr muss die Verarbeitung der Events implementiert werden.

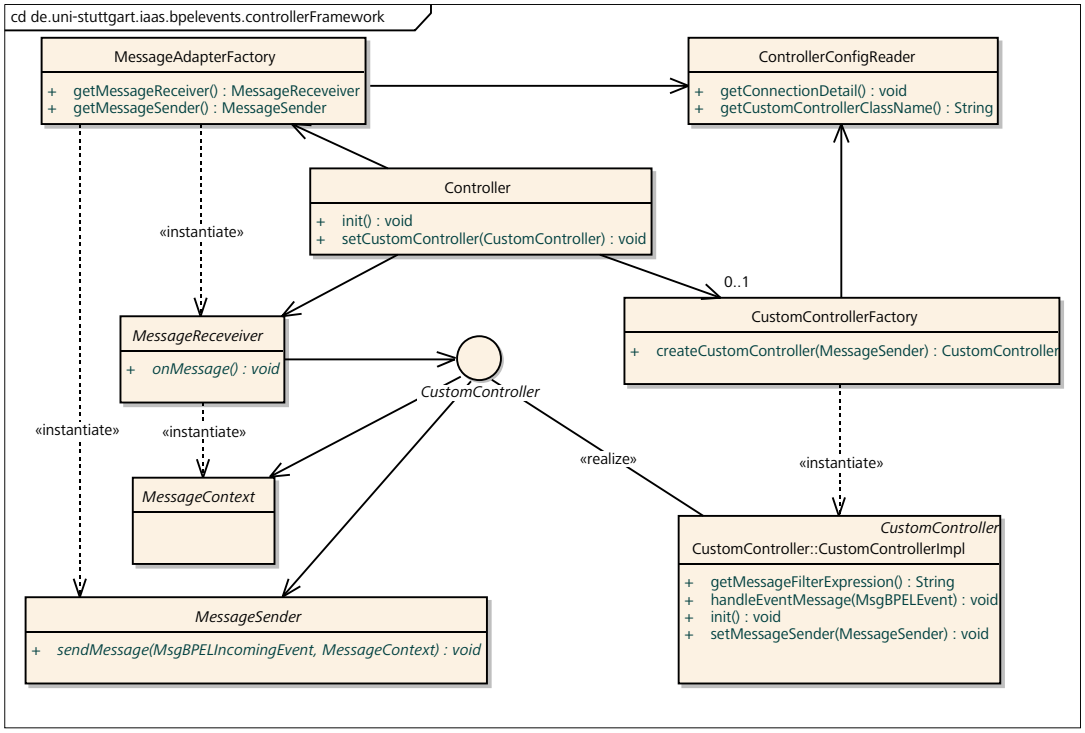


Abbildung 4.8: Entwurf des Controller Frameworks (JMS-spezifische Klassen wurden aus Gründen der Übersichtlichkeit weggelassen)

4.5 Aspekt Weaver

Der Weaver wird als Web-Applikation realisiert. Er stellt eine Web-Service Schnittstelle für das Deployment von Aspekten, eine Schnittstelle zum Event Framework und eine Messaging-Schnittstelle für Monitoring-Informationen (siehe [Wutke (2006)] und [Nitzsche (2006)]) bereit.

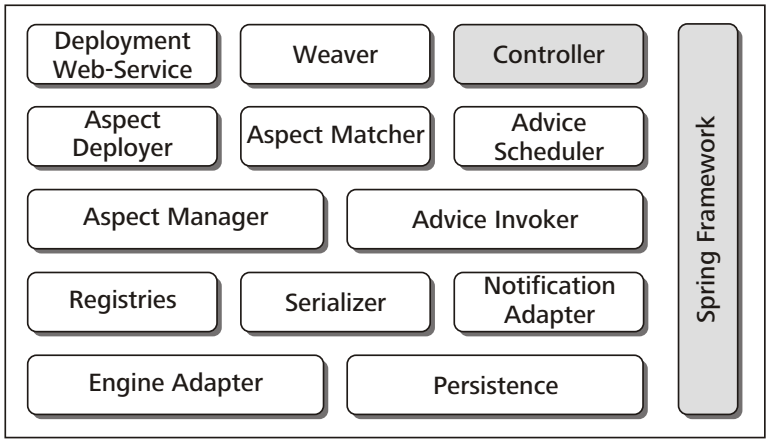


Abbildung 4.9: Schichtenarchitektur des Weavers (externe Komponenten sind grau hinterlegt)

Die Applikation besteht aus fünf Schichten, die aus mehreren Komponenten bestehen. In Abbildung 4.9 ist diese Schichtung dargestellt. Die Schichtung ist keine strenge Schichtung, da an einigen Stellen eine Schicht beim Zugriff übersprungen wird, wo dies sinnvoll erscheint. Es ist jedoch sichergestellt, dass keine Zugriffe von tiefer liegenden Schichten auf höhere erfolgen.

Abbildung 4.10 gibt die Komponenten-Architektur und die externen Schnittstellen detaillierter wieder.

Die hier dargestellten Komponenten werden auf Beans im Spring-Framework abgebildet. Teilweise bestehen sie noch aus weiteren Spring-Beans, die als Sub-Komponenten genutzt werden. Alle Komponenten sind in Form von Interfaces spezifiziert, so dass die Implementierungen durch eine Anpassung der Konfiguration ausgetauscht werden können, da die Spring-Konfiguration als eine Art Factory (vgl. [Gamma u. a. (1995)]) fungiert, die gemäß der Konfiguration die Komponenten instanziiert und untereinander verbindet. Die Konfiguration des Weavers erfolgt über die XML-Datei `applicationcontext.xml` sowie eine Properties-Datei, in der die Einstellungen des Weavers vorgenommen werden können.

Zur Initialisierung des Spring-Frameworks und damit des Weavers, wird eine Komponente des Spring-Frameworks, der `ContextLoaderListener`, verwendet, der genutzt wird um den `ApplicationContext` von Spring zu initialisieren.

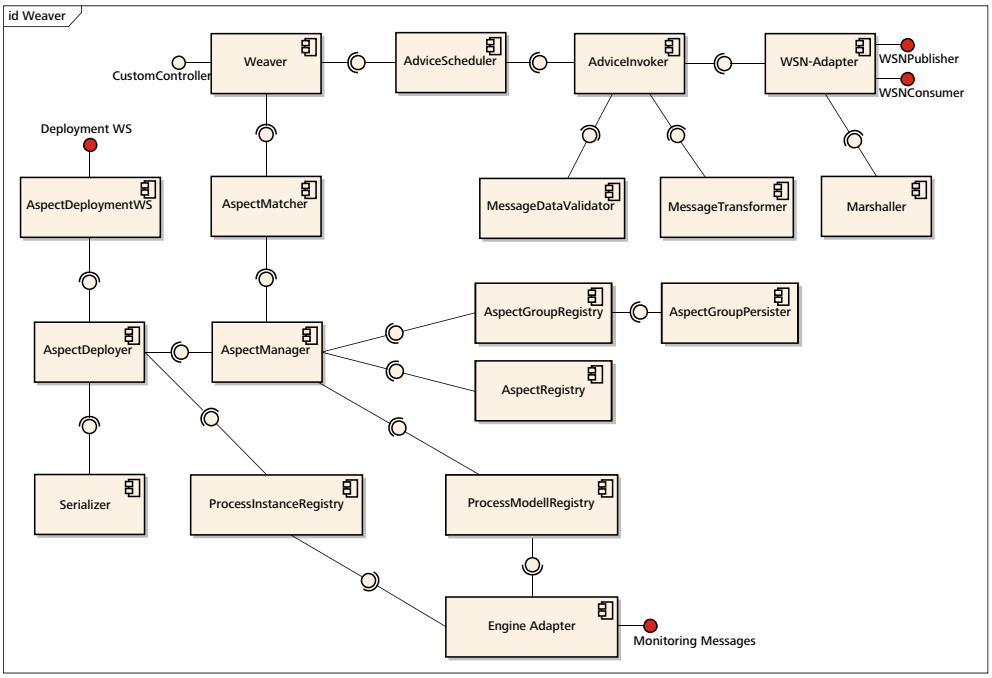


Abbildung 4.10: Interne Architektur des Weavers (externe Schnittstellen sind hervorgehoben)

Abbildung 4.11 zeigt die Interaktion der einzelnen Komponenten während des Weavings.

Im Folgenden werden die einzelnen Komponenten näher beschrieben.

4.5.1 Registries

Die *Registries* (vgl. [Fowler (2002)]) werden als zentraler Punkt zur Ablage und zum Zugriff auf Objekte genutzt. Die durch Registries verwalteten Objekte sind Prozessmodelle, Prozessinstanzen, Web-Service Endpunkte, Aspekte, Aspektgruppen und Topics. Im Folgenden werden die einzelnen Registries näher beschrieben.

Prozessmodell Registry Diese Registry speichert die in der BPEL-Engine deployten BPEL-Prozesse. Beim Deployment wird ein neues ProcessModel Objekt angelegt und unter dem qualifizierten Namen des Prozesses in die Registry aufgenommen. Beim Undeployment eines Prozesses wird das zugehörige Objekt aus der Registry entfernt.

Prozessinstanz Registry In der Prozessinstanz Registry werden die Informationen über einzelne Prozessinstanzen gespeichert. Bei der Instanziierung eines Prozessmodells oder beim Start des Weavers werden die Informationen über die Instanzen in der Registry gespeichert. Der Zugriff erfolgt über die engineweit eindeutige Prozessinstanz-ID (PIID).

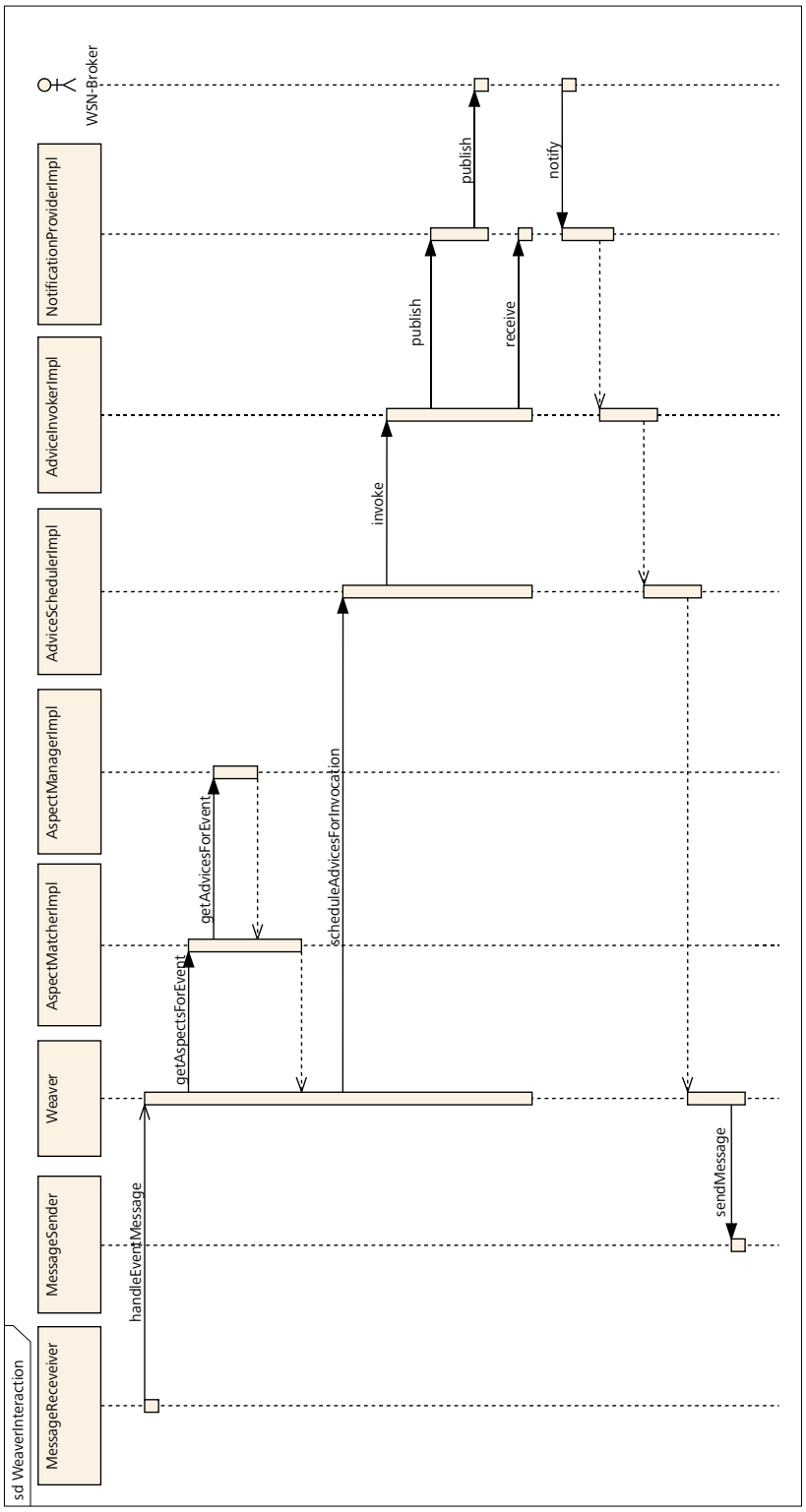


Abbildung 4.11: Ablauf des Weavings

Aspekt Registry Die Aspekt Registry speichert alle erfolgreich deployten Aspekte bis zu ihrem Undeployment. Der Zugriff auf die Aspekte erfolgt über ihren qualifizierten Namen.

Aspektgruppen Registry Die Verwaltung der erfolgreich deployten Aspektgruppen erfolgt durch die Aspektgruppen Registry. Sie erlaubt den Zugriff auf die registrierten Aspektgruppen über den qualifizierten Namen der Aspektgruppe.

Topic Registry Die Verwaltung der für die einzelnen Pointcuts definierten Topics übernimmt die Topic-Registry. Der Zugriff auf die Topics erfolgt über die Pointcut-Ausdrücke.

Zur persistenten Speicherung der Informationen über die Aspektgruppen wird die Komponente AspectGroupPersister eingesetzt. Sie bietet eine Schnittstelle, um den Inhalt einer AspectGroupRegistry persistent zu speichern. Die genutzte Implementierung verwendet einen Serializer (siehe Abschnitt 4.3.2) und speichert die Aspektgruppen im Dateisystem ab. Die Beziehungen zwischen den einzelnen Unterkomponenten ist in Abbildung 4.12 dargestellt.

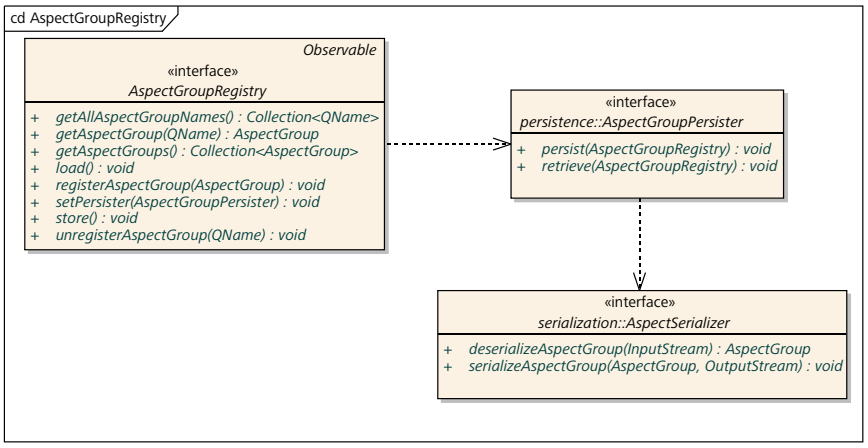


Abbildung 4.12: Struktur der Persistenz für Aspektgruppen

4.5.2 AspectManager

Der *AspectManager* ist für die Verwaltung der deployten Aspekten und die Bestimmung der für ein Event relevanten Advices zuständig. Hierzu löst er beim Deployment eines Aspekts die einzelnen Pointcuts in Mengen von Joinpoints innerhalb eines Prozessmodells auf. Hierbei werden die XPath-Ausdrücke, die zur Spezifikation der Pointcuts verwendet werden auf das BPEL-Dokument des Prozessmodells angewendet und in Mengen von kanonischen

Lokationspfaden (Location-Paths) aus dem BPEL-Dokument aufgelöst, die im weiteren Verlauf des Weavings genutzt werden, um die Joinpoints zu identifizieren.

Um einen effizienten Zugriff auf die einzelnen Aspekte anhand von Prozessmodell, Lokationspfad und Eventtyp zu erlauben, und damit eine hohe Performance des Weavers zu erreichen, wurde eine Datenstruktur gewählt, die wie folgt aufgebaut ist. Die Basis bildet eine Map, die über den qualifizierten Namen eines Prozessmodells indiziert ist. Sie enthält Referenzen auf weitere Maps, die über den Lokationspfad der Quelle eines Events innerhalb des Prozessmodells indiziert sind. Diese Maps wiederum halten Referenzen auf weitere Maps, die über den Typ von Events indiziert sind und Referenzen auf Listen von Advices, die die für einen Joinpoint relevanten Advices darstellen, enthalten. In Abbildung 4.13 wird diese Datenstruktur nochmals grafisch wiedergegeben.

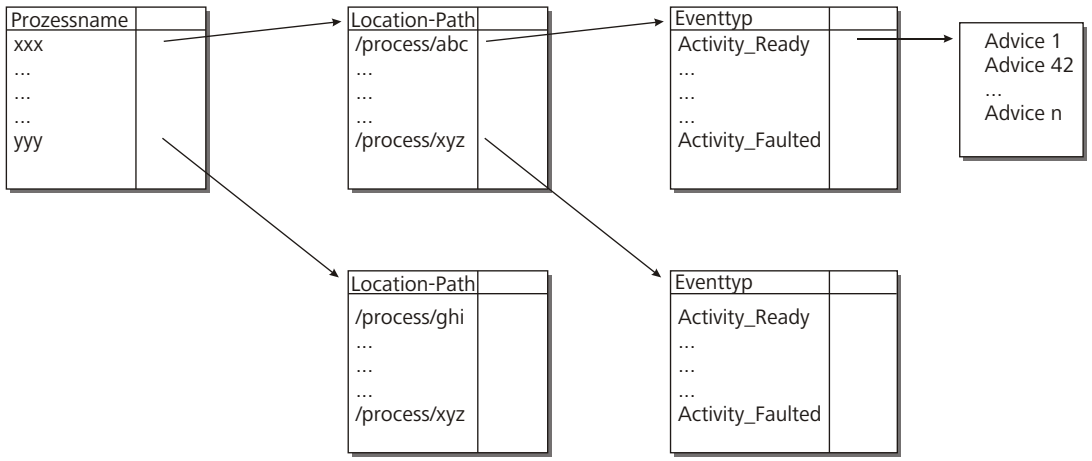


Abbildung 4.13: Aufbau der vom AspectManager verwendeten Datenstruktur

Die Überprüfung, ob ein Advice auf eine bestimmte Instanz angewendet werden soll, erfolgt nicht im AspectManager, sondern im *AspectMatcher* (siehe Abschnitt 4.5.3).

Der *AspectDeployer* registriert beim Deployment die Aspekte einer Aspektgruppe beim AspectManager. Dieser übernimmt die Prüfung, ob ein Aspekt direkt anwendbar ist, oder ob ein *Defered Deployment* (siehe Abschnitt 3.2.4.6) notwendig ist. Falls dies der Fall ist, wird der Aspekt in eine andere Datenstruktur eingefügt. Das eigentliche Einfügen von zurückgestellten Aspekten in die Datenstruktur der deployten Aspekte erfolgt, wenn das erste Event des Prozessmodells oder der Prozessinstanz, für die der Aspekt zurückgestellt wurde, vom AspectManager verarbeitet wird.

4.5.3 AspectMatcher

Der *AspectMatcher* ist für die Entscheidung, ob ein Advice für ein konkretes Event angewendet wird, verantwortlich. Die Implementierung des AspectMatchers nutzt für diesen Vergleich die Prozessinstanz-ID (PIID) bei Aspekten, die für bestimmte Instanzen deployt wurden. Zusätzlich wird für Aspekte, die für ganze Prozessmodelle deployt wurden überprüft, ob die aktuelle Instanz eine der Instanzen ist, die beim Deployment ausgeschlossen wurden, da für sie die „Alles-oder-Nichts"-Eigenschaft für die Aspektgruppe nicht gewährleistet werden konnte. Hierdurch werden die Advices bestimmt, die für ein Event wirklich angewendet werden müssen.

Diese Komponente ist in der momentanen Implementierung relativ klein und fügt zu der vom AspectManager gebotenen Funktionalität nicht viel hinzu. Der Grund für ihre Existenz ist jedoch, dass sie die Stelle ist, an der die in Abschnitt 3.2.4.5 beschriebene Identifikation von Instanzen anhand von Variableninhalten erfolgen kann.

Die Interaktion zwischen Weaver, AspectMatcher und AspectManager ist in Abbildung 4.14 nochmals dargestellt.

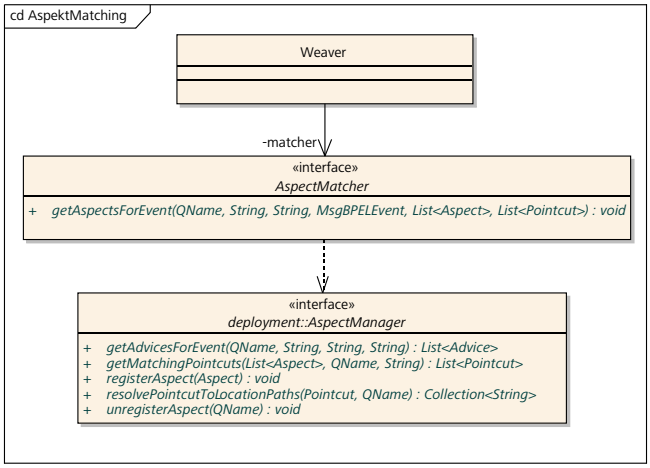


Abbildung 4.14: Interaktion von Weaver, AspectMatcher und AspectManager

4.5.4 AspectDeployer

Der *AspectDeployer* ist die Komponente, die für das Deployment von Aspekten verantwortlich ist. Er erhält serialisierte Aspektgruppen, deserialisiert sie, überprüft ihre Deploybarkeit, fügt sie in die Registries ein und übergibt sie dem AspectManager zur Registrierung. Die Überprüfung ob eine Aspektgruppe deployt werden kann übernimmt die Subkompo-

nente *AspectDeployabilityChecker*, der die in Abschnitt 3.2.4.6 beschriebenen Prüfungen ausführt.

4.5.5 Deployment Web-Service

Der *Aspekt Deployment Web-Service* bildet eine Web-Service Fassade (vgl. [Gamma u. a. (1995)]) für den AspectDeployer, die u. a. vom Aspekt Management Tool verwendet wird. Die Fassade wird über Apache Axis als Web-Service bereitgestellt. Um den Web-Service in das Spring-Framework zu integrieren und ihn konfigurieren zu können, erweitert die Fassade die Spring-Klasse `ServletEndpointSupport`.

Die Beziehungen zwischen den verschiedenen Komponenten beim Deployment sind in Abbildung 4.15 dargestellt.

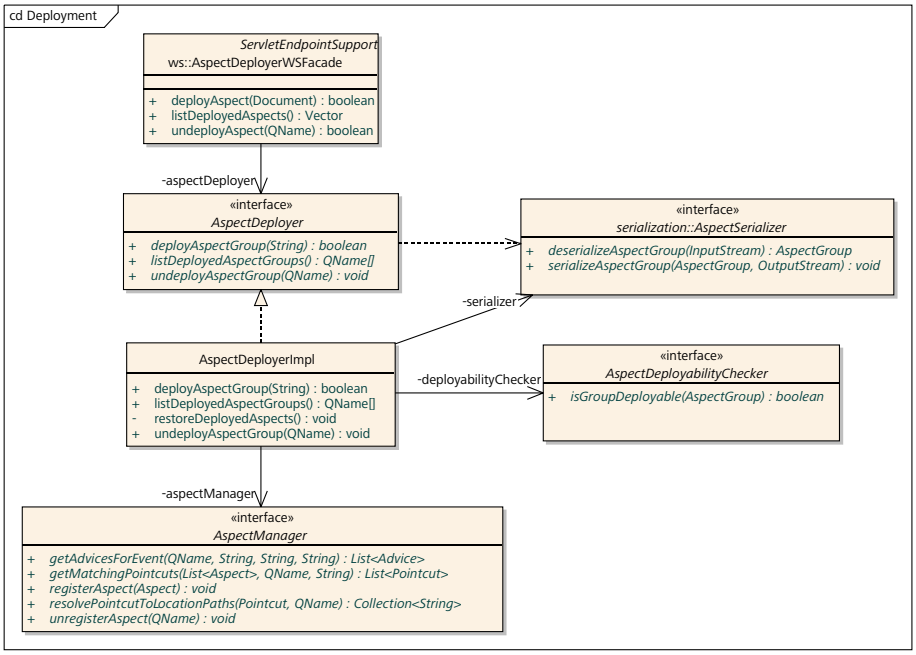


Abbildung 4.15: Struktur für das Deployment von Aspekten

4.5.6 AdviceScheduler

Der *AdviceScheduler* dient dazu, die in Abschnitt 3.2.4.8 beschriebenen Präzedenzregeln zwischen mehreren Advices herzustellen und die Handhabung der Parameter- und Rückga-

bewerte zu übernehmen. Die eigentlichen Aufrufe der Advices übergibt er an den AdviceInvoker.

4.5.7 AdviceInvoker

Der AdviceInvoker führt die Aufrufe der als Advice einzuwebenden Web-Services aus, indem er entsprechende InvocationRequest Nachrichten an den Wrapper erzeugt. Abbildung 4.16 zeigt die Beziehungen der am Aufruf beteiligten Komponenten.

Zunächst nutzt der AdviceInvoker beim Aufruf die *MessageValidator* Subkomponente, die sicherstellt, dass die vom AdviceScheduler übergebenen Parameterwerte zu den in der WSDL-Beschreibung des einzuwebenden Ports deklarierten Nachrichtenformaten passen. In der Implementierung erfolgt ausschließlich eine Prüfung des deklarierten Nachrichtentyps, keine Validierung der Nachricht gegen das deklarierte Schema.

Im Anschluss an die Validierung des Nachrichtentyps wird die optional spezifizierte Transformation der Nachricht durch die Komponente MessageTransformation ausgeführt. Hierzu wird die spezifizierte Transformation und die Nachricht an eine Factory (vgl. [Gamma u. a. (1995)]) übergeben, die eine Instanz einer Transformations-Strategy (vgl. [Gamma u. a. (1995)]) zurückgibt, die für die Transformation der Nachricht genutzt wird. Die Strategy bildet dabei die eigentliche *MessageTransformation* Komponente. Die Implementierung bietet zum einen die `VariableAccessTransformationStrategy` an, die auf Inhalte von Variablen, die durch ihren Namen spezifiziert werden, innerhalb der Prozessinstanz zugreift. Zum anderen wird die `NullTransformationStrategy` angeboten, die keinerlei Transformation ausführt und die unveränderte Nachricht zurückgibt.

Im Anschluss an die Transformation übergibt der AdviceInvoker die Transformierte Nachricht zusammen mit den Informationen über den einzuwebenden Endpunkt an den *NotificationProvider*.

Für die Speicherung der Rückgabewerte oder der aufgetretenen Fehler des eingewobenen Web-Services wird eine eigene Datenstruktur verwendet, die sich an den WSDL-Definitionen für Nachrichten und Faults orientiert. Sie ist in Abbildung 4.17 dargestellt.

4.5.8 Notification Adapter

Der *Notification Adapter* ist für den Versand von Invocation Request Nachrichten an den Wrapper und für den Empfang der Response Notification Nachrichten des Wrappers zuständig. Hierzu nutzt er die Marshaller-Komponente, die die Java-Objekte in XML-Dokumente

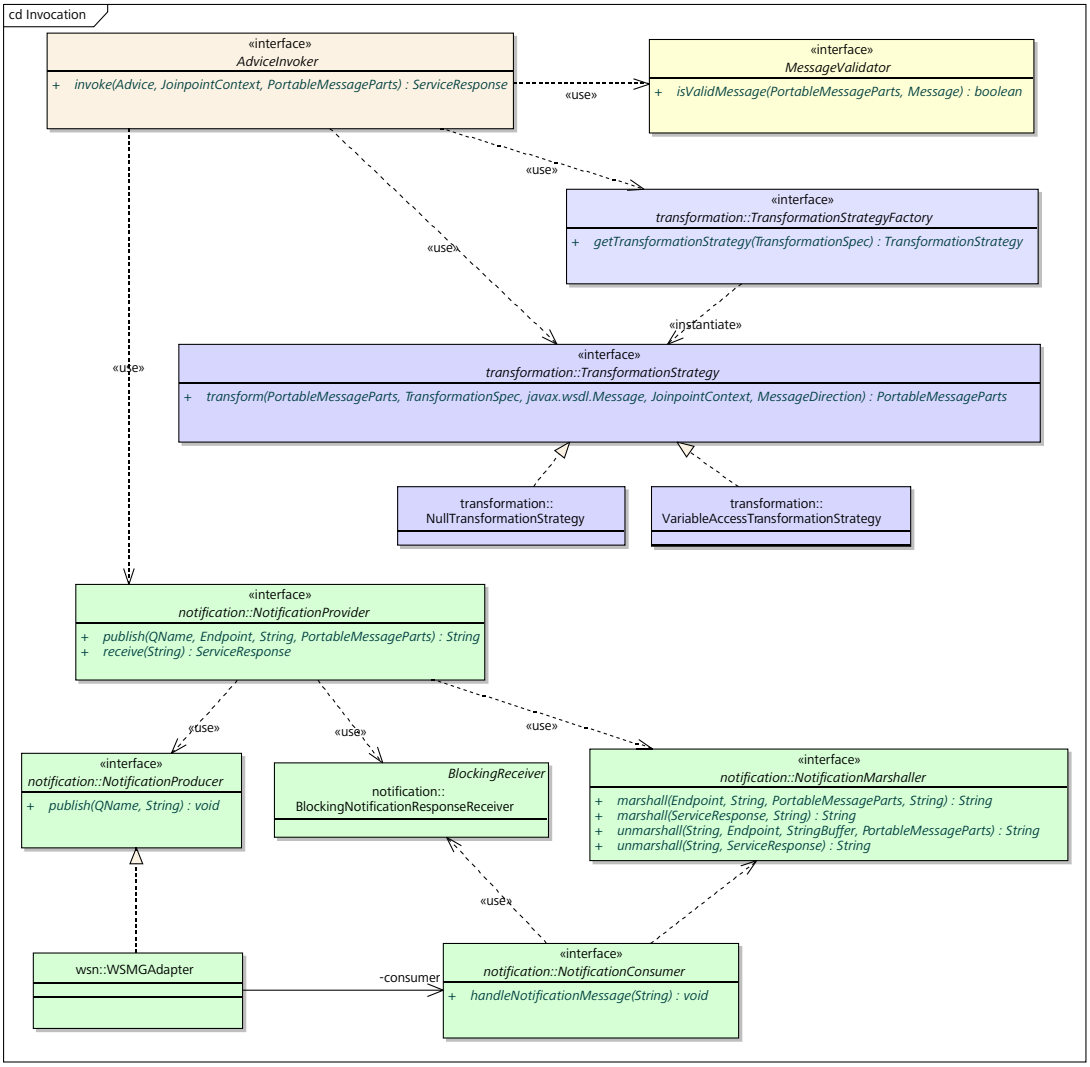


Abbildung 4.16: Struktur des Aufrufs von Advices

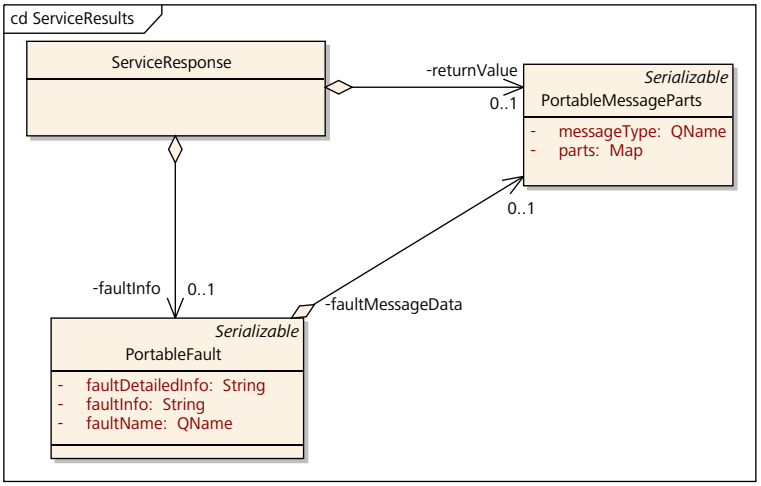


Abbildung 4.17: Datenstruktur für die Ablage der Resultate eines Web-Service Aufrufs

und umgekehrt umsetzt. Anschließend übergibt er das so erhaltene InvocationRequest-Dokument an den NotificationProducer, der die Nachricht an den WS-Notification Broker weitergibt. Im Anschluss daran führt der Notification Adapter einen Aufruf an den Blocking-Receiver auf, der solange blockiert, bis eine Response Notification Nachricht empfangen wurde, oder ein konfigurierbarer Timeout eintritt. Die Antworten werden von der NotificationConsumer Subkomponente empfangen, von ihr durch den Marshaller in Java-Objekte zurückübersetzt und anschließend an den BlockingReceiver übergeben, der sie an den NotificationAdapter zurückgibt.

4.5.9 Marshaller

Der *Marshaller* ist für die Umwandlung der Weaver-internen Objekte in ein für den Versand über den Notificationkanal geeignetes Format (Marshalling) und aus diesem Format zurück in Objekte (Unmarshalling) zuständig. Für den Versand über WS-Notification wird eine Serialisierung in ein XML-Format verwendet (in Abschnitt 3.2.4.12 wird dieses Format näher beschrieben). Für die Serialisierung kommt kein Databinding-Framework wie z. B. JAXB zum Einsatz, da diese i. d. R. aus einem XML-Schema Java-Klassen erzeugen, was aber mit dem vorhandenen Domänenmodell nicht vereinbar ist. Stattdessen wird eine eigene Implementierung auf Basis des DOM für das Marshalling und Unmarshalling verwendet.

Durch eine Änderung der Konfiguration des Spring-Frameworks lassen sich problemlos auch andere Serialisierungsformate durch andere Implementierungen des Marshallers nutzen.

4.5.10 EngineAdapter

Der *EngineAdapter* ist der einzige XEngine-spezifische Teil des Weavers. Er dient dazu den Zugriff auf Informationen der Engine wie deployte Prozessmodelle, Zustände von Instanzen oder Variableninhalte bereitzustellen. Hierzu bedient er sich des von [Wutke (2006)] und [Nitzsche (2006)] entwickelten Monitoring-Frameworks der XEngine. Er subskribiert sich auf die Monitoring-Messages und wertet diese aus.

Um Zugriff auf die Inhalte von Variablen innerhalb einer Instanz zu erlauben wurde es notwendig, den XEngine Controller (XEC) um eine neue Steuerungsnachricht zu erweitern. Dies geschah ebenso wie die in Abschnitt 4.4.1 beschriebene Integration des Event-Frameworks in die XEngine durch Nutzung von aspektorientierter Programmierung.

Durch die Nutzung von AOP ist diese Erweiterung ebenfalls zwischen verschiedenen Versionen der XEngine portabel und wurde sowohl mit Version 1.2 (Basierend auf ActiveBPEL 1.2) als auch mit Version 2.0 (basierend auf ActiveBPEL 2.0) erfolgreich angewendet. Allerdings gibt es eine Einschränkung in der Verwendung des Variablenzugriffsmechanismus. Bei einer Konfiguration der Engine für den persistenten Betrieb kommt es beim Zugriff auf Variablen zu Deadlocks. Der Grund hierfür ist, dass ActiveBPEL einen eigenen Mutex-Mechanismus verwendet und die Business Process Engine (bzw. genauer gesagt jeder Worker-Thread, der für die Verarbeitung einer Instanz zuständig ist) selbst einen Mutex auf jede Prozessinstanz hält, bis die Instanz abgeschlossen ist. Da die Verarbeitung der Messages durch den XEngine Controller jedoch in einem anderen Thread als die Business Process Engine erfolgt, kommt es zu den oben erwähnten Deadlocks.

4.6 Aspekt Management Tool

Das Aspekt Management Tool wird als Swing-Applikation realisiert. Die Applikation besteht aus drei Schichten (wenn man die interne Schichtung der Business-Logik-Schicht betrachtet, vier). Die einzelnen Schichten bestehen wiederum aus mehreren Komponenten (siehe Abbildung 4.18).

Für die Konfiguration des Aspekt Management Tools wird wie auch für den Weaver das Spring-Framework verwendet. Die Konfiguration des Spring-Frameworks erfolgt über die Datei `applicationcontext.xml` und eine zugehörige Properties-Datei.

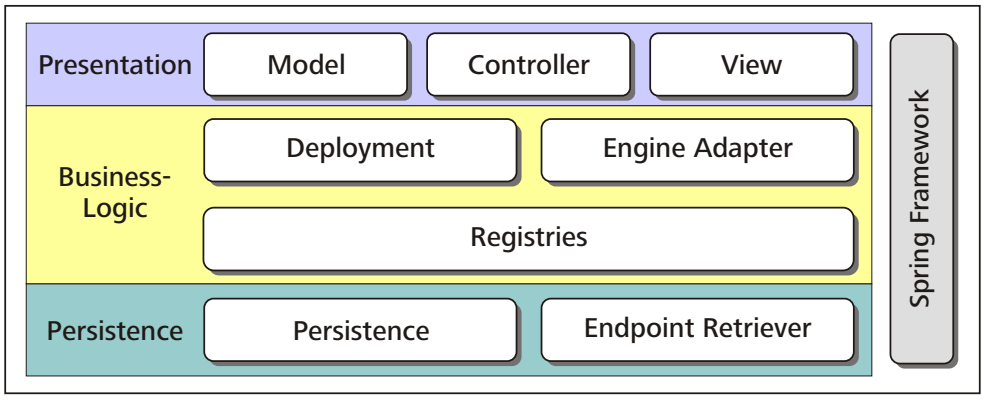


Abbildung 4.18: Schichtenarchitektur des Aspect Management Tools

4.6.1 Präsentationsschicht

Die *Präsentationsschicht* ist nach dem *Model-View-Controller* Paradigma (vgl. [Buschmann u. a. (1996)]) aufgebaut. Für jedes Fenster der Anwendung existiert eine eigene View-Klasse und ein zugehöriger Controller. Als GUI-Toolkit wurde Swing genutzt.

Für die Verbindung der Views und der zugehörigen Controller wird das Spring-Framework genutzt, so dass sich die Views austauschen lassen.

Die realisierte GUI bietet die in Abschnitt 3.2.6.1 beschriebenen Fenster.

4.6.2 Business-Logik-Schicht

4.6.2.1 Deployment Komponente

Die *Deployment* Komponente ist für das Deployment und Undeployment, sowie die Auflistung der deployten Aspektgruppen zuständig. Die Komponente stellt einen Proxy (vgl. [Gamma u. a. (1995)]) dar, der auf den Deployment Web-Service des Weavers (siehe Abschnitt 4.5.5 und Abbildung 4.1 auf Seite 78) zugreift.

4.6.2.2 Engine-Adapter

Der *Engine-Adapter* nutzt wie auch der Engine-Adapter des Weavers (siehe Abschnitt 4.5.10) das Monitoring Framework der XEngine. Er dient dazu, die momentan in der Engine

deployten Prozessmodelle festzustellen, um diese für das Deployment der Aspektgruppen als Ziel anzubieten.

4.6.2.3 Registries

Ebenso wie der Weaver verwendet das Aspekt Management Tool *Registries* (vgl. [Fowler (2002)]) als zentrale Stelle für die Ablage und den Zugriff auf Objekte. Hierfür werden die selben Implementierungen genutzt wie im Weaver (siehe Abschnitt 4.5.1). Sie werden jedoch teilweise anders konfiguriert, um beispielsweise ein anderes Seralisierungsformat für die persistente Speicherung der Aspektgruppen zu nutzen.

Aspektgruppen Registry Die Verwaltung der im Aspekt Management Tool verwalteten Aspektgruppen wird von der Aspektgruppen Registry übernommen. Sie erlaubt den Zugriff auf die registrierten Aspektgruppen über den qualifizierten Namen der Aspektgruppe.

Aspekt Registry Die Aspekt Registry speichert alle im Aspekt Management Tool verwalteten Aspekte. Der Zugriff auf die Aspekte erfolgt über ihren qualifizierten Namen.

Prozessmodell Registry Diese Registry speichert die in der BPEL-Engine deployten BPEL-Prozesse. Beim Deployment wird ein neues ProcessModel Objekt angelegt und unter dem qualifizierten Namen des Prozesses in die Registry aufgenommen. Beim Undeployment eines Prozesses wird das zugehörige Objekt aus der Registry entfernt.

Endpoint Registry Die Endpoint Registry enthält eine Liste aller bekannten Endpunkte, die als Ziel eines Advice genutzt werden können.

Die durch die *Endpoint Registry* verwalteten Informationen werden durch die Subkomponente *Endpoint Retriever* beim Start des Tools beschafft. Die in der Umsetzung verwendete Implementierung nutzt im Dateisystem abgelegte Dateien, die WS-Addressing Endpunktreferenzen enthalten, wie in Abschnitt 3.2.6.2 beschrieben. Für die Nutzung einer Service Registry wie beispielsweise einer UDDI-Registry, ist nur eine neue Implementierung dieser Subkomponente notwendig.

4.6.3 Persistenz-Schicht

Die Persistenz-Schicht des Aspekt Management Tools nutzt die in Abschnitt 4.3.2 beschriebenen Serialisierungskomponenten, um die verwalteten Aspektgruppen persistent im Dateisystem abzulegen. Es wird hierbei die Serialisierung in eine WS-Policy verwendet.

Die serialisierten Aspektgruppen werden in einem konfigurierbaren Verzeichnis abgelegt.

4.7 WS-Notification Wrapper

Der WS-Notification Wrapper wird als stand-alone Java-Applikation realisiert. Wie auch der Weaver und das Aspekt Management Tool besteht der Wrapper aus Komponenten, deren Wiring und Konfiguration das Spring-Framework übernimmt.

Abbildung 4.19 zeigt die Architektur des Wrappers.

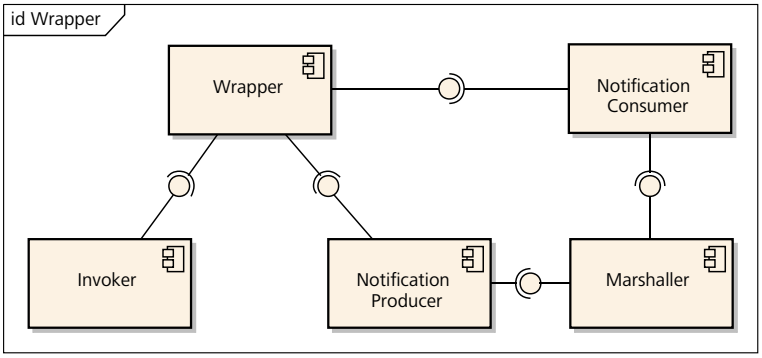


Abbildung 4.19: Architektur des WS-Notification Wrappers

4.7.1 Notification Consumer

Der *Notification Consumer* ist für den Empfang der InvocationRequest Nachrichten zuständig. Er wird vom WS-Notification Broker aufgerufen, wenn eine neue Nachricht für einen Topic, auf dem der Wrapper subskribiert ist, eintrifft. Um die Nachricht in ein Java-Objekt umzuwandeln nutzt er den *Marshaller* (siehe Abschnitt 4.5.9), der das Unmarshalling der Nachricht durchführt.

Die verarbeiteten Nachrichten werden über eine Callback-Funktion an die Wrapper-Komponente übergeben, die die weitere Bearbeitung übernimmt.

4.7.2 Wrapper

Die Implementierung des *Wrapper* ist die Hauptkomponente des WS-Notification Wrappers. Er initialisiert zum einen das Spring-Framework und damit die Applikation, zum anderen stellt er die Kernfunktionalität bereit.

Er übernimmt in einer Callback-Funktion die vom Notification Consumer empfangenen

Informationen des InvocationRequest, übergibt sie an den Invoker, der den einzuwebenden Web-Service aufruft und dessen Resultat (oder evtl. geworfene Faults) zurückgibt. Das Resultat übergibt der Wrapper dann dem Marshaller. Anschließend übergibt der Wrapper das gemarshallte Objekt an den Notification Producer, der es auf dem Result-Topic veröffentlicht.

4.7.3 Invoker

Der *Invoker* wird für die eigentlichen Aufrufe der einzuwebenden Web-Services genutzt. Er ist ein generischer Web-Service Client. Die für die Realisierung verwendete Implementierung stellt einen generischen SOAP-Client dar. Er unterstützt sowohl den Document/Literal als auch den RPC/Encoded Stil für die Aufrufe. Um die korrekte Zuordnung der Parameter und die richtige Auswahl des zu verwendenden Stils für den Aufruf des Web-Service sicherzustellen verwendet der generische SOAP-Client eine WSDL-Beschreibung des einzuwebenden Ports.

4.7.4 Notification Producer

Der *Notification Producer* ist für die Publikation der ResponseNotification Nachrichten auf den für die Antwortnachrichten festgelegten Topic zuständig. Die ResponseNotification Nachrichten wurden vor der Publikation vom Marshaller (siehe Abschnitt 4.5.9) aus den vom Invoker erzeugten `ServiceResponse` Objekten (siehe Abbildung 4.17 auf Seite 95) aufgebaut. Die für die Realisierung eingesetzte Implementierung übergibt die ResponseNotification Nachrichten an den WS-Notification Broker zur weiteren Veröffentlichung.

4.8 Beispielprozess

In diesem Abschnitt wird die Tauglichkeit des beschriebenen Lösungsansatzes anhand mehrerer Szenarien untersucht und mit anderen Lösungsansätzen verglichen.

Als Beispielprozess wird ein rudimentärer Bestellprozess eines Internet-Händlers verwendet. Er besteht nur aus den folgenden drei Schritten:

- Lieferung beim Logistikdienstleister beauftragen
- Gesamtpreis kalkulieren
- Abbuchung des Gesamtpreises veranlassen

Abbildung 4.20 zeigt den Beispielprozess als grafisches Prozessmodell, die ausführliche BPEL-Beschreibung des Prozesse findet sich im Anhang D in Listing D.1.

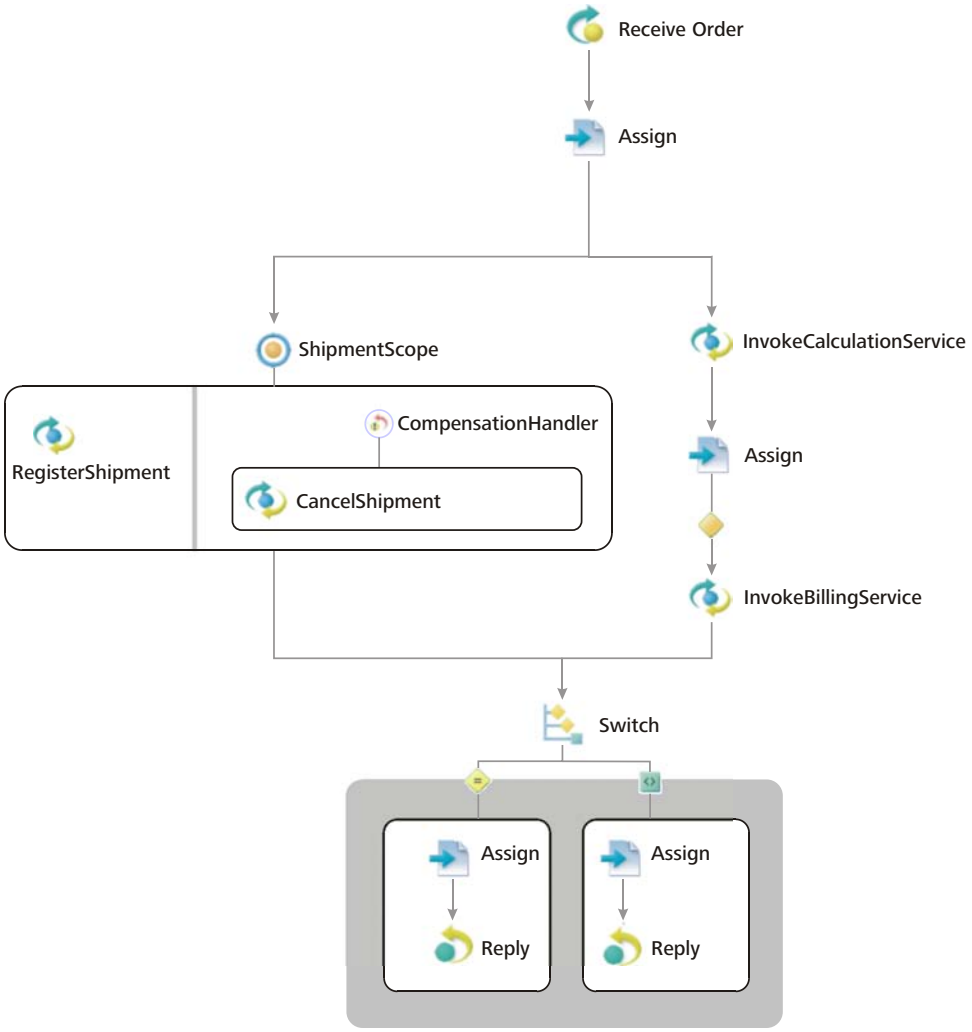


Abbildung 4.20: Struktur des Beispielprozesses

Für eine realistische Modellierung wäre u. a. auch für die Abbuchungs-Operation ein Kompensationshandler notwendig, die im hier verwendeten Beispiel jedoch aus Gründen der Übersichtlichkeit weggelassen wurde.

Die folgenden Szenarien werden betrachtet:

1. Der Händler möchte um das saisonale Geschäft anzukurbeln eine Weihnachtsaktion durchführen, bei der die Gesamtpreise jeder Bestellung pauschal um 5% gesenkt werden sollen. Eine Änderung des Kalkulationsservice möchte er jedoch nicht vornehmen.

2. Da einige Kunden in der Vergangenheit Probleme mit der Bezahlung hatten, möchte der Händler einen Dienst nutzen, der die Bonität des Kunden bei der Bestellung prüft. Um die Zuverlässigkeit (beispielsweise hinsichtlich Verfügbarkeit und Last) eines solchen Dienstleisters zu prüfen, möchte er jedoch vorerst keine Änderung seines Geschäftsprozesses vornehmen.
3. Um Missverständnissen bei der Abholung von Lieferungen durch den Logistikdienstleister zu vermeiden, möchte der Händler, dass falls eine Kompensation eines Abholauftrags stattfindet, zusätzlich eine e-Mail an die Versandabteilung gesendet wird.
4. Ein sehr guter Kunde hat eine größere Bestellung getätigt. Die Kundenbetreuungsabteilung möchte, dass diesem Kunden nach erfolgter Abrechnung eine Dank-eMail gesendet wird.
5. Der bisher genutzte Logistikdienstleister ist nicht mehr verfügbar und muss durch einen anderen ersetzt werden, der nicht zwingend denselben Porttyp implementiert wie der bisherige.

Im Folgenden wird die Umsetzung der einzelnen Szenarien mit Hilfe des hier entwickelten AOP-Mechanismus dargestellt.

4.8.1 Szenario: Weihnachtsrabatt

Die Berechnung des Weihnachtsrabatts lässt sich sehr einfach durch einen After-Advice abbilden, der das Resultat des Kalkulationsservice nutzt, um den rabattierten Rechnungsbetrag zu bestimmen. Hierzu wird ein spezieller Rabatt-Web-Service genutzt, der einen konfigurierbaren Prozentsatz oder Betrag (die Konfiguration des Web-Service Endpunkts wird hier nicht näher betrachtet) von der Rechnungssumme anzieht.

Listing 4.2 zeigt den relevanten Teil der Definition des Aspekts für die Umsetzung der Rabattaktion.

Listing 4.2: Ausschnitt aus der Definition des Aspekts für die Rabattaktion

```
<a4b:Aspect xmlns:a4b="http://www.iaas.uni-stuttgart.de/a4b"
            xmlns:ns1="http://mydotcom.com/services/order"
            Id="ns1:ChristmasDiscount">
  <a4b:Advice Name="DiscountAdvice">
    <a4b:When type="After"/>
    <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws/
                                      2004/08/addressing">
        <wsa:Address>
          http://localhost:8081/DemoServices/services/discountPort
        </wsa:Address>
        <wsa:PortType>ns1:discountPT</wsa:PortType>
```

```
12          <wsa:ServiceName wsa:PortName="discountPort">
13            ns1:discountService
14          </wsa:ServiceName>
15          <a4b:WSDLLocation>
16            http://localhost:8081/DemoServices/services/discountPort?wsdl
17          </a4b:WSDLLocation>
18        </wsa:EndpointReference>
19        <a4b:Operation name="calculateDiscountedPrice" SuppressFault="false"
20                       SuppressResult="false"/>
21      </a4b:Advice>
22      <a4b:Pointcut>
23        <a4b:ProcessArtifact type="activity"
24             identifier=
25               "/process/flow/invoke[@name='InvokeCalculationService']"/>
26      </a4b:Pointcut>
27    </a4b:Aspect>
```

Dieser Aspekt kann zu Beginn der Rabattaktion deployt und am Ende der Rabattaktion wieder undeployt werden. Weitere Schritte sind nicht notwendig.

Für die Realisierung dieses Szenarios ohne Nutzung von AOP gibt es mindestens zwei Ansätze:

1. Modifikation des Kalkulations-Service
2. Modifikation des Prozessmodells

Eine Modifikation des Kalkulations-Service würde einen vollständigen Durchlauf des Wartungszyklus für die geänderte Fassung zu Beginn der Rabattaktion, sowie eine Ersetzung durch die bestehende Version am Ende der Aktion bedeuten. Durch Nutzung eines Ansatzes wie dem von [Karastoyanova u. a. (2005b)] entwickelten, ließe sich ein Parallelbetrieb beider Versionen und ein dynamisches Umschalten zwischen beiden Versionen erreichen. Darüber hinaus entsteht bei diesem Ansatz jedoch potentiell der Bedarf, zwei Versionen des Kalkulations-Service zu warten, mit allen damit verbundenen negativen Implikationen, wenn während der Rabattaktion weitere Änderungen am Kalkulations-Service (man denke an wichtige Fehlerkorrekturen oder Änderungen des Mehrwertsteuersatzes etc.) notwendig werden sollten. Das Problem der Wartung mehrerer Versionen ließe sich jedoch durch Nutzung von AOP auf der Ebene der Implementierung des Service beheben. Aus den hier geschilderten Gründen ist dieser Ansatz nicht empfehlenswert.

Der Ansatz das Prozessmodell zu ändern, erlaubt beispielsweise auch die Nutzung eines Rabatt-Service wie oben beschrieben. Die Änderung des Prozessmodells bedeutet jedoch, dass nach dem Deployment der geänderten Version alle bereits laufenden und noch nicht abgeschlossenen Prozessinstanzen abgebrochen und neu gestartet werden müssen. Für die

hier genutzte, rudimentäre Version des Geschäftsprozesses ist dies weniger problematisch, da dieser Prozess keine langlaufenden Interaktionen enthält. Die Situation stellt sich für reale Bestellabwicklungsprozesse anders dar, da diese durchaus mehrere Tage oder Wochen laufen können, so dass ein Abbrechen und Neustarten dieser Prozesse nicht akzeptabel ist.

Man sieht, dass durch die Nutzung des AOP-Ansatzes in diesem Szenario Flexibilität gewonnen wird.

4.8.2 Szenario: Einführung der Bonitätsprüfung

Für die Einführung der Bonitätsprüfung soll der neue Dienstleister unter Realbedingungen evaluiert werden, zunächst jedoch so, dass seine Auskunft keinen Einfluss auf den Ablauf des Prozesses hat. Erst wenn der Dienst sich als verlässlich erwiesen hat, soll er in den Prozess integriert werden und die Bonitätsprüfung über die Annahme der Bestellung entscheiden.

Die Nutzung der Bonitätsprüfung lässt sich als OnLinkEvaluation-Advice realisieren, der den Wert des Links zwischen der Assign-Aktivität und der IvokeBillingService Aktivität ändert. Für die Testphase wird der Aspekt mit aktivierter Unterdrückung seines Resultats und evtl. aufgetretener Faults genutzt, wenn er in den Produktivbetrieb übernommen werden soll, wird die Unterdrückung deaktiviert, damit das Resultat des Web-Service als Wert der Transition-Condition in die Prozessinstanz zurückpropagiert wird.

Listing 4.3 zeigt die Definition des zugehörigen Aspekts. Er nutzt den Inhalt der Variable `billingRequest` als Parameter für den Bonitätsprüfungs-Service.

Listing 4.3: Ausschnitt aus der Definition des Aspekts für die Bonitätsprüfung

```
<a4b:Aspect xmlns:a4b="http://www.iaas.uni-stuttgart.de/a4b"
            xmlns:ns1="http://mydotcom.com/services/order"
            Id="ns1:RatingCheck">
  <a4b:Advice Name="RatingAspect">
    <a4b:When type="OnLinkEvaluation"/>
    <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws/
                                      2004/08/addressing">
      <wsa:Address>
        http://localhost:8081/DemoServices/services/creditworthinessPort
      </wsa:Address>
      <wsa:PortType>ns1:creditworthinessPT</wsa:PortType>
      <wsa:ServiceName wsa:PortName="creditworthinessPort">
        ns1:creditworthinessService
      </wsa:ServiceName>
      <a4b:WSDLLocation>
```

```
16          http://localhost:8081/DemoServices/services/creditworthinessPort
                ?wsdl
17        </a4b:WSDLLocation>
18      </wsa:EndpointReference>
19      <a4b:Operation name="isCreditworthy" SuppressFault="true"
            SuppressResult="true"/>
20      <a4b:InputTransformation>billingRequest</a4b:InputTransformation>
21    </a4b:Advice>
22    <a4b:Pointcut>
23      <a4b:ProcessArtifact type="link" identifier="L5"/>
24    </a4b:Pointcut>
25  </a4b:Aspect>
```

Dieser Aspekt kann zu Beginn der Testphase deployt und an deren Ende undeployt werden. Um ihn für den Produktivbetrieb einzusetzen, muss lediglich eine Änderung der beiden Attribute `SuppresResult` und `SuppressFault` in Zeile 19 vorgenommen und der Aspekt neu deployt werden.

Für die Realisierung dieses Szenarios ohne Rückgriff auf AOP Techniken ist eine (bzw. zwei, wenn man die Aktivierung der Bonitätsprüfung für den Produktivbetrieb berücksichtigt) Modifikation des Prozessmodells notwendig. Diese hat die in Abschnitt 4.8.1 beschriebenen Implikationen.

Ein Ansatz der, auf der dynamischen Änderung der genutzten Services basiert (siehe [Karastoyanova u. a. (2005b)]), ist ebenfalls denkbar. Er würde aber eine geänderte Fassung des Billing-Service erfordern, der vor dem eigentlichen Abbuchungsvorgang noch die Bonitätsprüfung durchführt. Dies ist zum einen nachteilig, was die Kopplung und den Zusammenhalt des Service angeht. Zum anderen ist diese Lösung nicht immer möglich, da für den Billing-Service möglicherweise ein externer Dienstleister genutzt wird.

Auch in diesem Szenario bietet der AOP-Ansatz erheblich mehr Flexibilität.

4.8.3 Szenario: Benachrichtigung der Versandabteilung

Dieses Szenario lässt sich durch Nutzung eines OnCompensation-Aspekts und eines passenden Web-Service, der eMails an die Versandabteilung generiert, realisieren.

Listing 4.4: Ausschnitt aus der Definition des Aspekts für die Benachrichtigung bei Kompensation

```
<a4b:Aspect xmlns:a4b="http://www.iaas.uni-stuttgart.de/a4b"
            xmlns:ns1="http://mydotcom.com/services/order"
            Id="ns1:CompensationNotificationAspect">
  <a4b:Advice Name="CompensationNotification">
    <a4b:When type="OnCompensation"/>
    <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws
        /2004/08/addressing">
      <wsa:Address>
        http://localhost:8081/DemoServices/services/
            emailNotificationPort
      </wsa:Address>
      <wsa:PortType>ns1:emailNotificationPT</wsa:PortType>
      <wsa:ServiceName wsa:PortName="emailNotificationPort">
        ns1:emailNotificationService
      </wsa:ServiceName>
      <a4b:WSDLLocation>
        http://localhost:8081/DemoServices/services/
            emailNotificationPort?wsdl
      </a4b:WSDLLocation>
    </wsa:EndpointReference>
    <a4b:Operation name="notifyShipmentDepartmentOfCompensation"
        SuppressFault="false" SuppressResult="false"/>
  </a4b:Advice>
  <a4b:Pointcut>
    <a4b:ProcessArtifact type="activity" identifier="/process/flow/scope
        [@name='ShipmentScope']"/>
  </a4b:Pointcut>
</a4b:Aspect>
```

Eine Realisierung dieses Szenarios ohne die Nutzung von AOP wäre ausschließlich durch Modifikation des Prozessmodells möglich, da der genutzte Web-Service zu einem externen Logistikdienstleister gehört. Für die Modifikation des Prozessmodells gelten die in Abschnitt 4.8.1 beschriebenen Implikationen.

4.8.4 Szenario: Dank-eMail an Kunden

Dieses Szenario erfordert eine instanzbasierten Veränderung des Geschäftsprozesses. Um den Einschränkungen der realisierten aspektorientierten BPEL-Engine Rechnung zu tragen wird hier vereinfachend angenommen, dass die Prozessinstanz-IDs der Bestellungen des guten Kunden bekannt sind. In der Realität müsste eine Identifikation anhand der Variableninhalte wie in Abschnitt 3.2.4.5 beschrieben erfolgen, was jedoch kein grundlegendes Problem darstellen sollte.

Um dieses Szenario zu realisieren bietet sich die Verwendung eines After-Advice an, der nach dem Aufruf des Billing-Service eingewoben wird. Er greift auf die Variable `orderRequest` zurück, um Informationen über den Kunden und seine Bestellung an den Web-Service übertragen zu können. In Listing 4.5 ist eine beispielhafte Aspektgruppe (hier wird eine vollständige Aspektgruppe dargestellt, da die Spezifikation einzelner Instanzen erst zur Deployment-Zeit durch den PolicyAttachment-Mechanismus erfolgen kann) zur Realisierung dieses Szenarios dargestellt. Es wird angenommen, dass die Prozessinstanz, die den Auftrag des wichtigen Kunden bearbeitet, die Instanz-ID 105 besitzt.

Listing 4.5: Definition des Aspekts für den Versand von Dank-eMails an wichtige Kunden

```
<wsp:PolicyAttachment
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:a4b="http://www.iaas.uni-stuttgart.de/a4b">
  <wsp:AppliesTo>
    <a4b:Processes>
      <a4b:Instance ID="105"/>
    </a4b:Processes>
  </wsp:AppliesTo>
  <wsp:Policy xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
      oasis-200401-wss-wssecurity-utility-1.0.xsd"
              wsu:Id="VIPCustomerAspects">
    <wsp:All>
      <a4b:Aspect xmlns:ns1="http://mydotcom.com/services/order"
                  Id="ns1:VIPCustomerMailAspect">
        <a4b:Advice Name="VIPMailSenderAdvice">
          <a4b:When type="Before"/>
          <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/
              ws/2004/08/addressing">
            <wsa:Address>
              http://localhost:8081/DemoServices/services/VIPMailPort
            </wsa:Address>
            <wsa:PortType>ns1:vipMailPT</wsa:PortType>
            <wsa:ServiceName wsa:PortName="VIPMailPort">
              ns1:VIPMailService
            </wsa:ServiceName>
            <a4b:WSDLLocation>
              http://localhost:8081/DemoServices/services/VIPMailPort?
                  wsdl
            </a4b:WSDLLocation>
          </wsa:EndpointReference>
          <a4b:Operation name="sendThankYouMail" SuppressFault="false"
                         SuppressResult="true"/>
          <a4b:InputTransformation>orderRequest</a4b:InputTransformation
              >
        </a4b:Advice>
        <a4b:Pointcut>
          <a4b:ProcessArtifact type="activity" identifier="/process/flow
```

```
                /invoke[@name='InvokeBillingService']"/>
34          </a4b:Pointcut>
35        </a4b:Aspect>
36      </wsp:All>
37    </wsp:Policy>
38  </wsp:PolicyAttachment>
```

Die Realisierung dieses Szenarios ohne Nutzung von AOP-Techniken erfordert wiederum eine Änderung des Prozessmodells. Dieses müsste um eine Fallunterscheidung, ob der Kunde einer Bestellung eine Dank-eMail erhalten soll, erweitert werden, was die Komplexität des Prozessmodells erhöht und die Wartbarkeit potentiell verringern kann. Darüber hinaus gelten zusätzlich die in Abschnitt 4.8.1 gemachten Aussagen zu den Nachteilen einer Änderung des Prozessmodells.

4.8.5 Szenario: Neuer Logistikpartner

Dieses Szenario lässt sich durch Nutzung zweier Instead-Advices lösen. Durch diese werden die Aufrufe für die Registrierung der Lieferung und deren Kompensation ersetzt. Um die Integrität des Prozesses sicherzustellen, müssen diese beiden Advices in einer gemeinsamen Aspektgruppe gruppiert werden, die für die notwendige Alles-oder-nichts Semantik sorgt.

Da die realisierte Implementierung keine echte Nachrichtentransformation unterstützt, wird im Folgenden nur ein Aspekt skizziert, der dieses Problem löst.

Listing 4.6: Ausschnitt aus der Definition des Aspekts für die Nutzung eines neuen Logistikpartners

```
1  <wsp:Policy xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
       oasis-200401-wss-wssecurity-utility-1.0.xsd"
2                wsu:Id="NewLogisticsAspects">
3    <wsp:All>
4      <a4b:Aspect xmlns:a4b="http://www.iaas.uni-stuttgart.de/a4b"
5                  xmlns:ns1="http://mydotcom.com/services/order"
6                  Id="ns1:ShipmentRegistrationAspect">
7        <a4b:Advice Name="ShipmentRegistration">
8          <a4b:When type="Instead"/>
9          <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws
               /2004/08/addressing">
10           ...
11         </wsa:EndpointReference>
12         <a4b:Operation name="comissionShipment" SuppressFault="false"
               SuppressResult="false"/>
13         <a4b:InputTransformation>
14           <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
                 Transform">
```

```
15          ...
16          </xsl:stylesheet>
17        </a4b:InputTransformation>
18        <a4b:OutputTransformation>
19          <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
                Transform">
20          ...
21          </xsl:stylesheet>
22        </a4b:OutputTransformation>
23      </a4b:Advice>
24      <a4b:Pointcut>
25        <a4b:ProcessArtifact type="activity"
26          identifier="/process/flow/scope[@name='ShipmentScope']/invoke[
                @name='RegisterShipment']"/>
27      </a4b:Pointcut>
28    </a4b:Aspect>
29    <a4b:Aspect xmlns:a4b="http://www.iaas.uni-stuttgart.de/a4b"
30                xmlns:ns1="http://mydotcom.com/services/order"
31                Id="ns1:ShipmentCancellationAspect">
32      <a4b:Advice Name="ShipmentCancellation">
33        <a4b:When type="Instead"/>
34        <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws
              /2004/08/addressing">
35          ...
36        </wsa:EndpointReference>
37        <a4b:Operation name="cancelComissionedShipment" SuppressFault="
              false" SuppressResult="false"/>
38        <a4b:InputTransformation>
39          <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
                Transform">
40          ...
41          </xsl:stylesheet>
42        </a4b:InputTransformation>
43      </a4b:Advice>
44      <a4b:Pointcut>
45        <a4b:ProcessArtifact type="activity"
46          identifier="/process/flow/scope[@name='ShipmentScope']/invoke[
                @name='CancelShipment']"/>
47      </a4b:Pointcut>
48    </a4b:Aspect>
49  </wsp:all>
50</wsp:policy>
```

Dieses Szenario lässt sich ohne Nutzung von AOP-Techniken beispielsweise durch Verwendung des von [Karastoyanova u. a. (2006)] beschriebenen Ansatz der parametrisierten Prozesse lösen, unter der Voraussetzung, dass eine semantische Beschreibung der beiden genutzten Logistikservices in einer Ontologie vorliegt. Eine andere Alternative wäre wie-

derum das Anhalten und Neustarten des Prozesses, was die in den vorherigen Abschnitten bereits beschriebenen Konsequenzen und Nachteile hat.

KAPITEL 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

In dieser Arbeit wurde ein Ansatz zur Nutzung von AOP-Techniken im Kontext von BPEL-Prozessen beschrieben. Die Grundlage dieser Arbeit bildet die Arbeit von [Karastoyanova u. a. (2007)], deren Ansätze verwendet wurden um eine Konzeption für eine AOP-fähige BPEL-Engine zu erstellen und einen prototypischen Aspekt-Weaver für BPEL zu realisieren.

Zunächst wurden die Eigenschaften von WS-Flows betrachtet und der Bedarf nach Möglichkeiten zur flexiblen Adaption für solche Prozesse beschrieben. Existierende Ansätze für die Flexibilität von BPEL-Prozesses wurden dargestellt und der Bedarf für weitergehende Flexibilitätsmechanismen gezeigt. Darauf aufbauend wurde betrachtet, welche Zielsetzungen mit der aspektorientierten Programmierung verfolgt werden und welche Eigenschaften aspektorientierte Ansätze aufweisen. Von diesen Eigenschaften ausgehend wurde dargestellt, wie AOP-Techniken für die flexible Adaption von BPEL-Prozessen genutzt werden können und welche Ansätze auf diesem Gebiet bereits existieren.

Im Rahmen des in dieser Arbeit beschriebenen Lösungsansatzes wurde gezeigt, wie sich die Konzepte der aspektorientierten Programmierung für die Anwendung auf BPEL-Prozesse adaptieren lassen und wie eine solche Adaption unter Nutzung von bestehenden Technologien aus dem Web-Service Umfeld, wie beispielsweise WS-Policy realisiert werden kann. Ausgehend davon wurde ein Konzept für die Realisierung eines Aspekt-Weavers für BPEL-Prozesse erarbeitet, der die in diesem Zusammenhang auftretenden Fragestellungen löst. Wo die Lösung über das Ziel dieser Arbeit hinausginge, wurden Lösungsansätze skizziert.

Für das Weaving wurde ein auf der Publikation der Navigationsevents einer BPEL-Engine aufbauender Ansatz gewählt, da dieser auf einer gemeinsamen Infrastruktur mit anderen Lösungen aufsetzen kann und die Verwendung von Events eine sehr gute Grundlage für

die Realisierung von aspektorientierten Mechanismen darstellt. Der Lösungsansatz nutzt einen zentralen Weaver, der als Broker die Navigationsevents verarbeitet, die durch einen Aspekt spezifiziert wurden, Aufrufe von Web-Services veranlasst und deren Resultate in die BPEL-Engine zurückgibt. Für den eigentlichen Aufruf der Web-Services wird eine über WS-Notification angebundene Wrapper-Komponente genutzt. Die Spezifikation und das Deployment der Aspekte erfolgt in einem einfachen grafischen Werkzeug.

Die im Lösungsansatz beschriebene Konzeption wurde in eine prototypische Implementierung umgesetzt, anhand derer die Tauglichkeit des Ansatzes evaluiert wurde. Hierzu wurden mehrere Szenarien betrachtet und die Nutzung von AOP und anderen Techniken zur Lösung der Fragestellung des Szenarios verglichen und gezeigt, dass die Nutzung von AOP einen höheren Grad an Flexibilität in diesen Szenarien bietet als andere Ansätze.

5.2 Ausblick

Während der Konzeption und Realisierung ergaben sich einige Fragestellungen für künftige Erweiterungen und Ergänzungen, die von Interesse für eine nähere Betrachtung sein dürften.

Ein Punkt, der für den produktiven Einsatz des hier beschriebenen AOP-Ansatzes von großer Bedeutung ist, ist die Identifikation von Instanzen, auf die ein Aspekt angewendet werden soll. Der im hier beschriebenen Lösungsansatz verwendete Weg, Instanzen anhand ihrer Prozessinstanz-ID (PIID) zu identifizieren, ist in der Praxis nicht gangbar, da es nicht möglich ist a priori zu wissen, welche ID eine Instanz von Interesse haben wird. Es ist deshalb notwendig, einen Mechanismus zu definieren, wie Instanzen anhand von Variableninhalten (beispielsweise des Correlation-Sets) o. ä. identifiziert werden können und wie ein solcher Identifikationsmechanismus auf die genutzten WS-Policies abgebildet werden kann. Ein möglicherweise tauglicher Ansatz auf diesem Gebiet wäre beispielsweise die Nutzung von XPath (vgl. [Clark und DeRose (1999)]). Eine weitere Fragestellung, die in diesem Themengebiet betrachtet werden muss, ist die Frage, wie mit Variableninhalten umgegangen wird. Variableninhalte können sich dynamisch ändern (im Fall der Nutzung von Correlation-Sets sind weniger Änderungen zu erwarten, aber durch die BPEL-Spezifikation nicht ausgeschlossen), so dass eine Überprüfung beim Deployment einer Aspektgruppe nicht ausreicht. In diesem Fall ist zu klären, ob und wie es überhaupt möglich ist, die „Alles-oder-Nichts“-Eigenschaft für Aspektgruppen zu garantieren.

Ein weiterer Punkt, der für den produktiven Einsatz von Bedeutung ist, ist die Möglichkeit der Transformation von Nachrichteninhalten, die zu und von den eingewobenen Web-Services gesendet werden. Ein solcher Mediationsmechanismus ist von Bedeutung, um die verschiedenen Web-Service-Schnittstellen einander anpassen zu können, insbesondere

um externe Web-Services als Advices benutzen zu können, die andere Nachrichtenformate nutzen als die im Prozessmodell vorhandenen. Es ist noch offen, auf welche Art solche eine Mediation ausgeführt werden kann und wie sie innerhalb der genutzten Policies spezifiziert werden kann. Ein naheliegender Ansatz ist die Nutzung von XSLT zur Spezifikation der Transformationen, jedoch gibt es möglicherweise noch weitere vorteilhafte Ansätze, wie z. B. semantische Beschreibungen.

Die Behandlung der Präzedenz von Aspekten ist momentan etwas rudimentär. Es wäre zu klären, wie es möglich ist, die Reihenfolge von Aspekten explizit zu definieren (beispielsweise über Prioritäten etc.) und wie die Datenübergabe zwischen mehreren Advices aussehen kann. Ebenso ist in diesem Zusammenhang ein Mechanismus wünschenswert, über den eine kontrollierte Übergabe von Parameterwerten und Resultaten zwischen Advices möglich wird.

Ebenso ist das Thema Auditing ein interessantes Themengebiet in diesem Bereich. Es ist noch zu klären, wie ein konkreter Auditing-Mechanismus für den in dieser Arbeit beschriebenen Lösungsansatz aussehen könnte und wie man die so gewonnenen Informationen mit dem Audit-Trail der BPEL-Engine verbinden kann.

Eng mit dem Thema Auditing verbunden ist das Thema Security. Es ist zu klären, auf welche Art und Weise sichergestellt werden kann, dass nur berechtigte Personen Aspekte einweben dürfen und dass nachvollziehbar ist, wer welchen Aspekt wann definiert, deployt oder undeployt hat. Das Thema Security erstreckt sich jedoch nicht nur über den Bereich des Auditing, sondern auch über die Nutzung sicherer Kommunikationskanäle zwischen Engine und Weaver, zwischen Weaver und Wrapper, sowie zwischen Wrapper und den einzuwebenden Web-Services. Hier ist zu betrachten, inwiefern WS-Security in diesem Zusammenhang genutzt werden kann.

Nicht direkt mit dem Thema Security verbunden, jedoch verwandt, ist die Fragestellung wie man mit QoS-Zusicherungen des zugrundeliegenden Prozesses wie Persistenz, Transaktionalität, Antwortzeiten usw. umgeht und wie man solche Zusicherung im Rahmen des AOP-Mechanismus unterstützen kann. Ein anderer Punkt, der in diesem Zusammenhang weiterer Betrachtung bedarf, ist die Frage, wie der Weaver zusammen mit der XEngine in einer persistenten Konfiguration genutzt werden kann. In der momentan realisierten Form ist dies aufgrund eines Sperrmechanismus in der XEngine nicht möglich, da hierbei Deadlocks auftreten können.

Für die Prüfung der Anwendbarkeit von Aspektgruppen (siehe Abschnitt 3.2.4.6) beim Neustart des Weavers bedarf es noch weiterer Überlegungen, um sicherzustellen, dass die „Alles-oder-Nichts“-Eigenschaft von Aspektgruppen auch in diesem Fall garantiert ist und keine Unterschiede zwischen dem Zustand vor und nach dem Neustart entstehen.

Ein weiterer Punkt, der eine sinnvolle Weiterentwicklung darstellen könnte, ist die automa-

tische Generierung der Deployment Deskriptoren für das engineseitige Event-Framework aus der Beschreibung von Aspekten.

Nicht zuletzt ist die Frage, ob und wie man den hier beschriebenen aspektorientierten Ansatz zur Flexibilität von WS-Flows noch flexibler gestalten kann, interessant. In diesem Zusammenhang ist es beispielsweise von Interesse zu betrachten, ob durch Nutzung des von [Karastoyanova u. a. (2005b, 2006)] beschriebenen Ansatzes der Parametrisierung bei der Anwendung von Aspekten eine weitere Flexibilisierung erreicht werden kann, indem man die einzuwebenden Web-Services beispielsweise mittels semantischer Ansätze selektiert, anstatt feste Web-Service Endpunkte vorzugeben und ob ein solcher Ansatz sinnvoll ist.

ANHANG A

Schema-Definitionen der Policy-Assertions

Listing A.1: Schema-Definitionen der Policy-Assertions

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.iaas.uni-stuttgart.de/projects/2006/a4b"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:wsa="http://schemas
    .xmlsoap.org/ws/2004/08/addressing" targetNamespace="http://www.iaas.
    uni-stuttgart.de/projects/2006/a4b">
  <xs:import namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing
      " schemaLocation="http://schemas.xmlsoap.org/ws/
                    2004/08/addressing/"/>
  <xs:complexType name="EndpointReferenceExtension">
    <xs:annotation>
      <xs:documentation>
        Extensibility Element for WSA-Endpoint-
        Reference to specifiy location of WSDL-Description
        associated with the specified Endpoint
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="WSDLLocation" type="xs:anyURI"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AspectType">
    <xs:sequence>
      <xs:element name="Advice" type="AdviceType"/>
      <xs:element name="Pointcut" type="PointcutType" maxOccurs="
          unbounded"/>
    </xs:sequence>
    <xs:attribute name="Id" type="xs:string" use="optional"/>
  </xs:complexType>
  <xs:complexType name="AdviceType">
    <xs:sequence>
      <xs:element name="When">
        <xs:complexType>
```

```
28          <xs:attribute name="Type" use="required">
29            <xs:simpleType>
30              <xs:restriction base="xs:string">
31                <xs:enumeration value="before"/>
32                <xs:enumeration value="instead"/>
33                <xs:enumeration value="after"/>
34                <xs:enumeration value="onFault"/>
35                <xs:enumeration value="onCompensation"/>
36              </xs:restriction>
37            </xs:simpleType>
38          </xs:attribute>
39        </xs:complexType>
40      </xs:element>
41      <xs:element ref="wsa:EndpointReference"/>
42      <xs:element name="Operation">
43        <xs:complexType>
44          <xs:attribute name="name" type="xs:string" use="required"/>
45        </xs:complexType>
46      </xs:element>
47      <xs:element name="InputTransformation" type="TransformationType"
            minOccurs="0"/>
48      <xs:element name="OutputTransformation" type="TransformationType"
            minOccurs="0"/>
49    </xs:sequence>
50    <xs:attribute name="name" type="xs:string" use="optional"/>
51    <xs:attribute name="suppressfault" type="xs:boolean" use="optional"/
        >
52    <xs:attribute name="suppressresult" type="xs:boolean" use="optional"
        />
53  </xs:complexType>
54  <xs:complexType name="PointcutType">
55    <xs:sequence>
56      <xs:element name="ProcessArtifact">
57        <xs:complexType>
58          <xs:attribute name="type" use="required">
59            <xs:simpleType>
60              <xs:restriction base="xs:string">
61                <xs:enumeration value="activity"/>
62                <xs:enumeration value="link"/>
63              </xs:restriction>
64            </xs:simpleType>
65          </xs:attribute>
66          <xs:attribute name="identifier" type="xs:string" use="required
              "/>
67        </xs:complexType>
68      </xs:element>
69    </xs:sequence>
70  </xs:complexType>
```

```
71    <xs:complexType name="TransformationType">
72      <xs:sequence>
73        <xs:any namespace="##other"/>
74      </xs:sequence>
75    </xs:complexType>
76    <xs:element name="Aspect" type="AspectType">
77      <xs:annotation>
78        <xs:documentation>
79          Element Describing an Aspect in A4B
80        </xs:documentation>
81      </xs:annotation>
82    </xs:element>
83    <xs:complexType name="ProcessesExpressionType">
84      <xs:sequence>
85        <xs:element name="Model" minOccurs="0">
86          <xs:complexType>
87            <xs:attribute name="name" type="xs:QName" use="required"/>
88          </xs:complexType>
89        </xs:element>
90        <xs:element name="Instance" minOccurs="0" maxOccurs="unbounded">
91          <xs:complexType>
92            <xs:attribute name="ID" type="xs:string" use="required"/>
93          </xs:complexType>
94        </xs:element>
95      </xs:sequence>
96    </xs:complexType>
97    <xs:element name="Processes" type="ProcessesExpressionType">
98      <xs:annotation>
99        <xs:documentation>
100         Element describing a Domain-Expression (BPEL-based)
101         to which processes an aspect should be applied
102       </xs:documentation>
103     </xs:annotation>
104   </xs:element>
105   <xs:element name="WSDLLocation" type="xs:anyURI"/>
106 </xs:schema>
```

ANHANG B

Schema-Definitionen der Notification Nachrichten

Listing B.1: Schema der Notification Nachrichten

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns="http://www.iaas.uni-stuttgart.de/projects/2006/
    a4b-notification" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    targetNamespace="http://www.iaas.uni-stuttgart.de/projects/2006/
    a4b-notification">
  <xs:import namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing
      " schemaLocation="http://schemas.xmlsoap.org/ws/2004/08/addressing/
      "/>
  <xs:complexType name="NotificationType">
    <xs:sequence>
      <xs:element name="CorrelationData" type="xs:string"/>
      <xs:element name="MessageData">
        <xs:complexType>
          <xs:sequence>
            <xs:any namespace="##other"/>
          </xs:sequence>
          <xs:attribute name="messageType" type="xs:QName" use="required
              "/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="InvocationNotificationType">
    <xs:complexContent>
      <xs:extension base="NotificationType">
        <xs:sequence>
          <xs:element ref="wsa:EndpointReference"/>
          <xs:element name="Operation">
            <xs:complexType>
              <xs:attribute name="name" type="xs:string" use="required"/
                  >
```

```
25            </xs:complexType>
26          </xs:element>
27        </xs:sequence>
28      </xs:extension>
29    </xs:complexContent>
30  </xs:complexType>
31  <xs:element name="InvocationNotification" type="
      InvocationNotificationType">
32    <xs:annotation>
33      <xs:documentation>Element describing message for invocation
          notification</xs:documentation>
34    </xs:annotation>
35  </xs:element>
36  <xs:complexType name="ResponseNotificationType">
37    <xs:complexContent>
38      <xs:extension base="NotificationType">
39        <xs:sequence>
40          <xs:element name="FaultData" minOccurs="0">
41            <xs:complexType>
42              <xs:sequence>
43                <xs:element name="FaultName" fixed="xs:QName"/>
44                <xs:element name="FaultInfo" type="xs:string"/>
45                <xs:element name="FaultDetail" type="xs:string"/>
46                <xs:element name="FaultMessageData">
47                  <xs:complexType>
48                    <xs:sequence>
49                      <xs:any namespace="##other"/>
50                    </xs:sequence>
51                  </xs:complexType>
52                </xs:element>
53              </xs:sequence>
54            </xs:complexType>
55          </xs:element>
56        </xs:sequence>
57      </xs:extension>
58    </xs:complexContent>
59  </xs:complexType>
60  <xs:element name="ResponseNotification" type="ResponseNotificationType
      ">
61    <xs:annotation>
62      <xs:documentation>Element describing message for invocation
          response notification</xs:documentation>
63    </xs:annotation>
64  </xs:element>
65</xs:schema>
```

ANHANG C

Schema-Definitionen der Event Deployment Deskriptoren

Listing C.1: Schema der Event Deployment Deskriptoren

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.iaas.uni-stuttgart.de/XEngine/
            EventDeploymentData">
        <xs:complexType name="EventDefinition">
                <xs:sequence>
                        <xs:element name="Node" type="xs:string" />
                        <xs:element name="Event" type="xs:string"
                            minOccurs="1" maxOccurs="unbounded"/>
                        <xs:element name="Blocking" type="xs:boolean" />
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="Definitions">
                <xs:sequence>
                        <xs:element name="EventDefinition" type="
                            EventDefinition"
                                minOccurs="1" maxOccurs="unbounded">
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="DeploymentData" type="Definitions" />
</xs:schema>
```

ANHANG D

Beispielprozess

Listing D.1: BPEL-Beschreibung des Beispielprozesses

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <process enableInstanceCompensation="yes" expressionLanguage="http://www
       .w3.org/TR/1999/REC-xpath-19991116" name="Demoprocess"
       suppressJoinFailure="yes" targetNamespace="http://mydotcom.com/
       services/order" xmlns="http://schemas.xmlsoap.org/ws/2003/03/
       business-process/" xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/
       business-process/" xmlns:ns1="Invalid Document" xmlns:ns2="http://
       mydotcom.com/services/order" xmlns:xsd="http://www.w3.org/2001/
       XMLSchema">
3    <partnerLinks>
4      <partnerLink myRole="OrderService" name="OrderServicePartnerLink"
           partnerLinkType="ns2:OrderServiceLinkType"/>
5      <partnerLink name="ShippingServicePartnerLink" partnerLinkType="
           ns2:ShippingServiceLinkType" partnerRole="ShippingService"/>
6      <partnerLink name="CalculationServicePartnerLink" partnerLinkType="
           ns2:calculationServiceLinkType" partnerRole="calculationService"/
           >
7      <partnerLink name="BillingServicePartnerLink" partnerLinkType="
           ns2:billingServiceLinkType" partnerRole="billingService"/>
8    </partnerLinks>
9    <variables>
10     <variable messageType="ns2:orderRequest" name="orderRequest"/>
11     <variable messageType="ns2:orderResponse" name="orderResponse"/>
12     <variable messageType="ns2:calculationRequest" name="
           calculationRequest"/>
13     <variable messageType="ns2:calculationResponse" name="
           calculationResponse"/>
14     <variable messageType="ns2:shippingRequest" name="shippingRequest"/>
15     <variable messageType="ns2:shippingResponse" name="shippingResponse"
           />
16     <variable messageType="ns2:billingRequest" name="billingRequest"/>
17     <variable messageType="ns2:billingResponse" name="billingResponse"/>
```

```
18    </variables>
19    <flow>
20      <links>
21        <link name="L1"/>
22        <link name="L7"/>
23        <link name="L2"/>
24        <link name="L3"/>
25        <link name="L4"/>
26        <link name="L5"/>
27        <link name="L6"/>
28      </links>
29      <receive createInstance="yes" name="ReceiveOrder" operation="order"
            partnerLink="OrderServicePartnerLink" portType="ns2:orderPT"
            variable="orderRequest">
30        <source linkName="L1"/>
31      </receive>
32      <scope name="ShipmentScope">
33        <target linkName="L2"/>
34        <source linkName="L7"/>
35        <invoke inputVariable="shippingRequest" name="RegisterShipment"
              operation="registerShipment" outputVariable="shippingResponse"
              partnerLink="ShippingServicePartnerLink" portType="
              ns2:shippingPT">
36        </invoke>
37        <compensationHandler>
38          <invoke inputVariable="shippingRequest" name="CancelShipment"
                operation="cancelShipment" partnerLink="
                ShippingServicePartnerLink" portType="ns2:shippingPT">
39            </invoke>
40        </compensationHandler>
41      </scope>
42      <assign>
43        <target linkName="L1"/>
44        <source linkName="L2"/>
45        <source linkName="L3"/>
46        <copy>
47          <from part="orderRequest" query="/orderRequest/item" variable="
                orderRequest"/>
48          <to part="calculationRequest" variable="calculationRequest"/>
49        </copy>
50        <copy>
51          <from part="orderRequest" query="/orderRequest/orderID" variable
                ="orderRequest"/>
52          <to part="orderID" variable="shippingRequest"/>
53        </copy>
54        <copy>
55          <from part="orderRequest" query="/orderRequest/deliveryAddress"
                variable="orderRequest"/>
```

```
56          <to part="address" variable="shippingRequest"/>
57        </copy>
58      </assign>
59      <invoke inputVariable="calculationRequest" name="
            InvokeCalculationService" operation="calculateBill"
            outputVariable="calculationResponse" partnerLink="
            CalculationServicePartnerLink" portType="ns2:calculationPT">
60        <target linkName="L3"/>
61        <source linkName="L4"/>
62      </invoke>
63      <assign>
64        <target linkName="L4"/>
65        <source linkName="L5" transitionCondition="true()"/>
66        <copy>
67          <from part="orderRequest" query="/orderRequest/customerID"
                variable="orderRequest"/>
68          <to part="customer" variable="billingRequest"/>
69        </copy>
70        <copy>
71          <from part="calculationResponse" variable="calculationResponse"/
                >
72          <to part="amount" variable="billingRequest"/>
73        </copy>
74        <copy>
75          <from expression="false()"/>
76          <to part="billingResponse" variable="billingResponse"/>
77        </copy>
78      </assign>
79      <invoke inputVariable="billingRequest" name="InvokeBillingService"
            operation="billCustomer" outputVariable="billingResponse"
            partnerLink="BillingServicePartnerLink" portType="ns2:billingPT">
80        <target linkName="L5"/>
81        <source linkName="L6"/>
82      </invoke>
83      <switch>
84        <target linkName="L6"/>
85        <target linkName="L7"/>
86        <case condition="(bpws:getVariableData('shippingResponse', '
              shippingResponse') and bpws:getVariableData('billingResponse',
              'billingResponse')) = true()">
87          <flow>
88            <links>
89              <link name="L8"/>
90            </links>
91            <assign>
92              <source linkName="L8"/>
93              <copy>
```

```
94                <from expression="concat(&quot;Order accepted. Charged
                      amount: &quot;, string( bpws:getVariableData('
                      calculationResponse', 'calculationResponse') ) )"/>
95                <to part="orderApproved" variable="orderResponse"/>
96              </copy>
97            </assign>
98            <reply operation="order" partnerLink="OrderServicePartnerLink"
                   portType="ns2:orderPT" variable="orderResponse">
99              <target linkName="L8"/>
100           </reply>
101         </flow>
102       </case>
103       <otherwise>
104         <flow>
105           <links>
106             <link name="L9"/>
107           </links>
108           <reply operation="order" partnerLink="OrderServicePartnerLink"
                   portType="ns2:orderPT" variable="orderResponse">
109             <target linkName="L9"/>
110           </reply>
111           <assign>
112             <source linkName="L9"/>
113             <copy>
114               <from>Order rejected.</from>
115               <to part="orderApproved" variable="orderResponse"/>
116             </copy>
117           </assign>
118         </flow>
119       </otherwise>
120     </switch>
121   </flow>
122 </process>
```

Listing D.2: WSDL-Definitionen der im Beispielprozess verwendeten Web-Services

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/
       business-process/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soapenc="
       http://schemas.xmlsoap.org/soap/encoding/" xmlns:plnk="http://schemas
       .xmlsoap.org/ws/2003/05/partner-link/" xmlns:xsd="http://www.w3.org
       /2001/XMLSchema" xmlns:y="http://mydotcom.com/services/order"
       xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://
       schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/
       wsdl/" targetNamespace="http://mydotcom.com/services/order">
3    <types>
4      <xs:schema targetNamespace="http://mydotcom.com/services/order"
```

```
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:simpleType name="orderResultType">
        <xs:restriction base="xs:string"/>
      </xs:simpleType>
      <xs:complexType name="addressType">
        <xs:sequence>
          <xs:element name="street" type="xs:string"/>
          <xs:element name="zipcode" type="xs:string"/>
          <xs:element name="city" type="xs:string"/>
          <xs:element name="country" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="orderItemType">
        <xs:sequence>
          <xs:element name="articleID" type="xs:string"/>
          <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="orderType">
        <xs:sequence>
          <xs:element name="orderID" type="xs:string"/>
          <xs:element name="item" type="y:orderItemType"/>
          <xs:element name="customerID" type="xs:string"/>
          <xs:element name="customerName" type="xs:string"/>
          <xs:element name="deliveryAddress" type="y:addressType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:simpleType name="shipmentRegistrationResultType">
        <xs:restriction base="xs:boolean"/>
      </xs:simpleType>
      <xs:simpleType name="invoiceAmountType">
        <xs:restriction base="xs:float"/>
      </xs:simpleType>
      <xs:simpleType name="billingResultType">
        <xs:restriction base="xs:boolean"/>
      </xs:simpleType>
    </xs:schema>
  </types>
  <message name="orderResponse">
    <part name="orderApproved" type="y:orderResultType"/>
  </message>
  <message name="orderRequest">
    <part name="orderRequest" type="y:orderType"/>
  </message>
  <message name="shippingResponse">
    <part name="shippingResponse" type="y:shipmentRegistrationResultType
        "/>
  </message>
```

```
51  <message name="calculationRequest">
52    <part name="calculationRequest" type="y:orderItemType"/>
53  </message>
54  <message name="discountRequest">
55    <part name="undiscountedPrice" type="y:invoiceAmountType"/>
56  </message>
57  <message name="calculationResponse">
58    <part name="calculationResponse" type="y:invoiceAmountType"/>
59  </message>
60  <message name="shippingRequest">
61    <part name="orderID" type="xs:string"/>
62    <part name="address" type="y:addressType"/>
63  </message>
64  <message name="billingRequest">
65    <part name="customer" type="xs:string"/>
66    <part name="amount" type="y:invoiceAmountType"/>
67  </message>
68  <message name="billingResponse">
69    <part name="billingResponse" type="y:billingResultType"/>
70  </message>
71  <message name="creditworthinessRequest">
72    <part name="customerID" type="xs:string"/>
73  </message>
74  <message name="creditworthinessResponse">
75    <part name="result" type="xs:boolean"/>
76  </message>
77  <message name="NewMessage"/>
78  <portType name="orderPT">
79    <operation name="order">
80      <input message="y:orderRequest"/>
81      <output message="y:orderResponse"/>
82    </operation>
83  </portType>
84  <portType name="shippingPT">
85    <operation name="cancelShipment">
86      <input message="y:shippingRequest"/>
87    </operation>
88    <operation name="registerShipment">
89      <input message="y:shippingRequest"/>
90      <output message="y:shippingResponse"/>
91    </operation>
92  </portType>
93  <portType name="calculationPT">
94    <operation name="calculateBill">
95      <input message="y:calculationRequest"/>
96      <output message="y:calculationResponse"/>
97    </operation>
98  </portType>
```

```
99  <portType name="discountPT">
100   <operation name="calculateDiscountedPrice">
101     <input message="y:discountRequest"/>
102     <output message="y:calculationResponse"/>
103   </operation>
104 </portType>
105 <portType name="billingPT">
106   <operation name="billCustomer">
107     <input message="y:billingRequest"/>
108     <output message="y:billingResponse"/>
109   </operation>
110 </portType>
111 <portType name="creditworthinessPT">
112   <operation name="isCreditworthy">
113     <input message="y:creditworthinessRequest"/>
114     <output message="y:creditworthinessResponse"/>
115   </operation>
116 </portType>
117 <portType name="emailNotificationPT">
118   <operation name="notifyShipmentDepartmentOfCompensation">
119     <input message="y:shippingRequest"/>
120   </operation>
121 </portType>
122 <portType name="vipMailPT">
123   <operation name="sendThankYouMail">
124     <input message="y:orderRequest"/>
125   </operation>
126 </portType>
127 <binding name="discountServiceSOAPBinding" type="y:discountPT">
128   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap
        /http"/>
129   <operation name="calculateDiscountedPrice">
130     <soap:operation soapAction="urn:#calculateDiscountedPrice"/>
131     <input>
132       <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
            use="encoded"/>
133     </input>
134     <output>
135       <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
            use="encoded"/>
136     </output>
137   </operation>
138 </binding>
139 <binding name="orderServiceSOAPBinding" type="y:orderPT">
140   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap
        /http"/>
141   <operation name="order">
142     <soap:operation soapAction="urn:#order"/>
```

```
143       <input>
144         <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              use="literal"/>
145       </input>
146       <output>
147         <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              use="literal"/>
148       </output>
149     </operation>
150   </binding>
151   <binding name="calculationServiceSOAPBinding" type="y:calculationPT">
152     <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap
          /http"/>
153     <operation name="calculateBill">
154       <soap:operation soapAction="urn:#calculateBill"/>
155       <input>
156         <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              use="encoded"/>
157       </input>
158       <output>
159         <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              use="encoded"/>
160       </output>
161     </operation>
162   </binding>
163   <binding name="shippingServiceSOAPBinding" type="y:shippingPT">
164     <soap:binding style="document" transport="http://schemas.xmlsoap.org
          /soap/http"/>
165     <operation name="cancelShipment">
166       <soap:operation soapAction="urn:#cancelShipment"/>
167       <input>
168         <soap:body use="literal"/>
169       </input>
170     </operation>
171     <operation name="registerShipment">
172       <soap:operation soapAction="urn:#registerShipment"/>
173       <input>
174         <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              use="literal"/>
175       </input>
176       <output>
177         <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              use="literal"/>
178       </output>
179     </operation>
180   </binding>
181   <binding name="billingServiceSOAPBinding" type="y:billingPT">
182     <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap
```

```
      /http"/>
183     <operation name="billCustomer">
184       <soap:operation soapAction="urn:#billCustomer"/>
185       <input>
186         <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              use="encoded"/>
187       </input>
188       <output>
189         <soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              use="encoded"/>
190       </output>
191     </operation>
192   </binding>
193   <binding name="creditworthinessServiceSOAPBinding" type="
        y:creditworthinessPT">
194     <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap
        /http"/>
195     <operation name="isCreditworthy">
196       <soap:operation soapAction="urn:#isCreditworthy"/>
197       <input>
198         <soap:body use="encoded"/>
199       </input>
200       <output>
201         <soap:body use="encoded"/>
202       </output>
203     </operation>
204   </binding>
205   <binding name="emailNotificationSOAPBinding" type="
        y:emailNotificationPT">
206     <soap:binding style="document" transport="http://schemas.xmlsoap.org
        /soap/http"/>
207     <operation name="notifyShipmentDepartmentOfCompensation">
208       <soap:operation soapAction="urn:#
            notifyShipmentDepartmentOfCompensation"/>
209       <input>
210         <soap:body use="literal"/>
211       </input>
212     </operation>
213   </binding>
214   <binding name="vipMailSOAPBinding" type="y:vipMailPT">
215     <soap:binding style="document" transport="http://schemas.xmlsoap.org
        /soap/http"/>
216     <operation name="sendThankYouMail">
217       <soap:operation soapAction="urn:#sendThankYouMail"/>
218       <input>
219         <soap:body use="literal"/>
220       </input>
221     </operation>
```

```
222   </binding>
223   <service name="orderService">
224     <port name="orderPort" binding="y:orderServiceSOAPBinding">
225       <soap:address location="http://localhost:8080/TestServices/order"
              xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"/>
226     </port>
227   </service>
228   <service name="calculationService">
229     <port name="calculationPort" binding="
          y:calculationServiceSOAPBinding">
230       <soap:address location="http://localhost:8081/DemoServices/
              services/calculationPort" xmlns:soap="http://schemas.xmlsoap.
              org/wsdl/soap/"/>
231     </port>
232   </service>
233   <service name="discountService">
234     <port name="discountPort" binding="y:discountServiceSOAPBinding">
235       <soap:address location="http://localhost:8080/TestServices/
              discount" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"/>
236     </port>
237   </service>
238   <service name="shippingService">
239     <port name="shippingPort" binding="y:shippingServiceSOAPBinding">
240       <soap:address location="http://localhost:8081/DemoServices/
              services/shippingPort" xmlns:soap="http://schemas.xmlsoap.org/
              wsdl/soap/"/>
241     </port>
242   </service>
243   <service name="billingService">
244     <port name="billingPort" binding="y:billingServiceSOAPBinding">
245       <soap:address location="http://localhost:8081/DemoServices/
              services/billingPort"/>
246     </port>
247   </service>
248   <service name="creditworthinessService">
249     <port name="creditworthinessPort" binding="
          y:creditworthinessServiceSOAPBinding">
250       <soap:address location="http://localhost:8081/TestServices/
              crediworthiness"/>
251     </port>
252   </service>
253   <service name="emailNotificationService">
254     <port name="emailNotificationPort" binding="
          y:emailNotificationSOAPBinding">
255       <soap:address location="No Target Adress"/>
256     </port>
257   </service>
258   <service name="VIPMailService">
```

```
259     <port name="VIPMailPort" binding="y:vipMailSOAPBinding">
260       <soap:address location="No Target Adress"/>
261     </port>
262   </service>
263   <plnk:partnerLinkType name="OrderServiceLinkType">
264     <plnk:role name="OrderService">
265       <plnk:portType name="y:orderPT"/>
266     </plnk:role>
267   </plnk:partnerLinkType>
268   <plnk:partnerLinkType name="ShippingServiceLinkType">
269     <plnk:role name="ShippingService">
270       <plnk:portType name="y:shippingPT"/>
271     </plnk:role>
272   </plnk:partnerLinkType>
273   <plnk:partnerLinkType name="calculationServiceLinkType">
274     <plnk:role name="calculationService">
275       <plnk:portType name="y:calculationPT"/>
276     </plnk:role>
277   </plnk:partnerLinkType>
278   <plnk:partnerLinkType name="billingServiceLinkType">
279     <plnk:role name="billingService">
280       <plnk:portType name="y:billingPT"/>
281     </plnk:role>
282   </plnk:partnerLinkType>
283   <bpws:property name="orderID" type="xsd:string"/>
284 </definitions>
```

Literaturverzeichnis

Alonso u. a. 2004
Alonso, Gustavo ; Casati, Fabio ; Kuno, Harumi ; Machiraju, Vijay: *Web Services - Concepts, Architectures and Applications*. Springer-Verlag, Berlin, Heidelberg, 2004

Bachmendo und Unland 2001
Bachmendo, Boris ; Unland, Rainer: Aspect-Based Workflow Evolution. In: Rashid, Awais (Hrsg.): *Tutorial and Workshop on Aspect-Oriented Programming and Separation of Concerns*, August 2001

Bajaj u. a. 2006a
Bajaj, Siddharth u. a.: Web Services Policy Attachment (WS-PolicyAttachment), Version 1.2 / IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, VeriSign. URL http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polatt/ws-polat-2006-03-01.pdf, März 2006. – Spezifikation

Bajaj u. a. 2006b
Bajaj, Siddharth u. a.: Web Services Policy Framework (WS-Policy), Version 1.2 / IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, VeriSign. URL http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polfram/ws-policy-2006-03-01.pdf, März 2006. – Spezifikation

Baresi und Guinea 2005
Baresi, Luciano ; Guinea, Sam: Towards Dynamic Monitoring of WS-BPEL Processes. In: Benatallah, Boualem (Hrsg.) ; Casati, Fabio (Hrsg.) ; Traverso, Paolo (Hrsg.): *Proceedings of the Third International Conference on Service-Oriented Computing - ICSOC 2005*. Amsterdam, The Netherlands : Springer, Dezember 2005 (Lecture Notes in Computer Science Bd. 3826), S. 269–282

Bellwood u. a. 2004
Bellwood, Tom ; Clement, Luc ; von Riegen, Claus u. a.: UDDI Version 3.0.2 /

OASIS. URL http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf, Oktober 2004. – Spezifikation

Booth u. a. 2004
Booth, David ; Haas, Hugo ; McCabe, Francis ; Newcomer, Eric ; Champion, Michael ; Ferris, Chris ; Orchard, David: Web Services Architecture / W3C. URL http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, Februar 2004. – W3C Working Group Note

Box u. a. 2004a
Box, Don u. a.: Web Services Addressing (WS-Addressing) / W3C. URL http://www.w3.org/Submission/ws-addressing/, August 2004. – Spezifikation

Box u. a. 2004b
Box, Don u. a.: Web Services Eventing (WS-Eventing) / OASIS. URL ftp://www6.software.ibm.com/software/developer/library/ws-eventing/WS-Eventing.pdf, August 2004. – Spezifikation

Box u. a. 2002
Box, Don ; Hondo, Maryann ; Kaler, Chris ; Maruyama, Hiroshi ; Nadalin, Anthony ; Nagaratnam, Nataraj ; Patrick, Paul ; von Riegen, Claus ; Shewchuk, John: Web Services Policy Assertions Language (WS-PolicyAssertions), Version 1.0 / IBM, BEA Systems, Microsoft, SAP AG. URL ftp://www6.software.ibm.com/software/developer/library/ws-polas.pdf, Dezember 2002. – Spezifikation

Burbeck 2000
Burbeck, Steve: The Tao of e-business services / IBM Software Group. URL http://www-128.ibm.com/developerworks/webservices/library/ws-tao/, Oktober 2000. – Technical Report

Buschmann u. a. 1996
Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons, 1996

Casati u. a. 1996
Casati, Fabio ; Ceri, Stefano ; Pernici, Barbara ; Pozzi, Giuseppe: Workflow Evolution. In: *International Conference on Conceptual Modeling / the Entity Relationship Approach*, 1996, S. 438–455

Chappell 2004
Chappell, David: *Enterprise Service Bus: Theory in Practice*. O'Reilly Media, 2004

Charfi und Mezini 2004
Charfi, Anis ; Mezini, Mira: Aspect-Oriented Web Service Composition with AO4BPEL. In: Zhang, Liang-Jie (Hrsg.) ; Jeckle, Mario (Hrsg.): *Web Services, European Conference, ECOWS 2004, Erfurt, Germany, September 27-30, 2004, Proceedings*, Springer-Verlag, Berlin, Heidelberg, 2004 (Lecture Notes in Computer Science Bd. 3250), S. 168–182

Charfi und Mezini 2005a
Charfi, Anis ; Mezini, Mira: An aspect-based process container for BPEL. In: *AOMD '05: Proceedings of the 1st workshop on Aspect oriented middleware development*. New York, NY, USA : ACM Press, 2005

Charfi und Mezini 2005b
Charfi, Anis ; Mezini, Mira: Using Aspects for Security Engineering of Web Service Compositions. In: *Proceedings of the 2005 IEEE International Conference on Web Services, Volume I*, Juli 2005, S. 59–66

Chavez und Lucena 2003
Chavez, Christina von Flach G. ; Lucena, Carlos José P.: A Theory of Aspects for Aspect-oriented Software Development. In: *Proceedings of the XVII. Brazilian Symposium on Software Engineering*. Manaus, Brazil, Oktober 2003

Cilia u. a. 2003
Cilia, Mariano ; Haupt, Michael ; Mezini, Mira ; Buchmann, Alejandro P.: The Convergence of AOP and Active Databases: Towards Reactive Middleware. In: *Proceedings of the International Conference on Generative Programming and Component Engineering (GPEC'03)*, September 2003 (Lecture Notes in Computer Science Bd. 2830), S. 169–188

Clark und DeRose 1999
Clark, James ; DeRose, Steve: XML Path Language (XPath), Version 1.0 / W3C. November 1999. – Spezifikation

Courbis und Finkelstein 2004
Courbis, Carine ; Finkelstein, Anthony: Towards an Aspect Weaving BPEL engine. In: Coady, Y. (Hrsg.) ; Lorenz, D. H. (Hrsg.): *Proceedings of the Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS)*. Lancaster, United Kingdom, März 2004

Courbis und Finkelstein 2005a
Courbis, Carine ; Finkelstein, Anthony: Towards Aspect Weaving Applications. In: *ICSE '05: Proceedings of the 27th international conference on Software engineering*. New York, NY, USA : ACM Press, 2005, S. 69–77

Courbis und Finkelstein 2005b
Courbis, Carine ; Finkelstein, Anthony: Weaving Aspects into Web Service Orchestrations. In: *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*. Washington, DC, USA : IEEE Computer Society, 2005, S. 219–226

Curbera u. a. 2003
Curbera, Francisco ; Goland, Yaron ; Klein, Johannes ; Leymann, Frank ; Roller, Dieter ; Thatte, Satish ; Weerawarana, Sanjiva: Business Process Execution Language for Web Services (BPEL4WS), Version 1.1 / BEA, IBM, Microsoft, SAP AG and Siebel Systems. URL http://www.ibm.com/developerworks/library/ws-bpel, Mai 2003. – Spezifikation

Deshpande 2007
Deshpande, Rajkumar: *Erweiterung einer existierenden Workflows-Engine um eine Auditing-Komponente*, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, Diplomarbeit, 2007. – noch nicht erschienen

Elrad u. a. 2001
Elrad, Tzilla ; Filman, Robert E. ; Bader, Atef: Aspect-oriented programming: Introduction. In: *Communications of the ACM* 44 (2001), Oktober, Nr. 10, S. 29–32

Erradi u. a. 2005
Erradi, Abdelkarim ; Maheshwari, Piyush ; Padmanabhuni, Srinivas: Towards a Policy-Driven Framework for Adaptive Web Services Composition. In: *Proceedings of the International Conference on Next Generation Web Services Practices (NWeSP'05)*. Los Alamitos, CA, USA : IEEE Computer Society, 2005, S. 33–38

Eugster u. a. 2003
Eugster, Patrick T. ; Felber, Pascal A. ; Guerraoui, Rachid ; Kermarrec, Anne-Marie: The many faces of publish/subscribe. In: *ACM Computing Surveys* 35 (2003), Nr. 2, S. 114–131

Fowler 2002
Fowler, Martin: *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002

Fowler 2004
Fowler, Martin: *Inversion of Control Containers and the Dependency Injection pattern*. Januar 2004. – URL http://martinfowler.com/articles/injection.html

Gamma u. a. 1995
Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series)*. Addison-Wesley Professional, 1995

Geppert und Tombros 1998
Geppert, Andreas ; Tombros, Dimitrios: Event-based Distributed Workflow Execution with EVE. In: Davies, N. (Hrsg.) ; Raymond, K. (Hrsg.) ; Seitz, J. (Hrsg.): *Proceedings IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing Middleware '98*, Springer, 1998

Gosling u. a. 2005
Gosling, James ; Joy, Bill ; Steele, Guy ; Bracha, Gilad: *The Java Language Specification*. Third Edition. Addison-Wesley, Mai 2005

Gottschalk 2000
Gottschalk, Karl: Web Services Architecture Overview / IBM developerWorks. URL http://www-128.ibm.com/developerworks/web/library/w-ovr/, September 2000. – Artikel

Gradecki und Lesiecki 2003
Gradecki, Joseph D. ; Lesiecki, Nicholas: *Mastering AspectJ: Aspect-Oriented Programming in Java*. New York, NY, USA : John Wiley & Sons, Inc., 2003

Graham u. a. 2004a
Graham, Steve ; Niblett, Peter u. a.: Publish-Subscribe Notification for Web services, Version 1.0 / OASIS. URL http://www-128.ibm.com/developerworks/library/ws-pubsub/WS-PubSub.pdf, März 2004. – Whitepaper

Graham u. a. 2004b
Graham, Steve ; Niblett, Peter u. a.: Web Service Base Notification (WS-Base Notification), Version 1.0 / OASIS. URL ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BaseN.pdf, März 2004. – Spezifikation

Graham u. a. 2004c
Graham, Steve ; Niblett, Peter u. a.: Web Service Brokered Notification (WS-BrokeredNotification), Version 1.0 / OASIS. URL ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BrokeredN.pdf, März 2004. – Spezifikation

Graham u. a. 2004d
Graham, Steve ; Niblett, Peter u. a.: Web Services Topics (WS-Topics), Version 1.0 / OASIS. URL ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-Topics.pdf, März 2004. – Spezifikation

Han u. a. 1998
Han, Yanbo ; Sheth, Amit ; Bussler, Christoph: A Taxonomy of Adaptive Workflow Management. In: *Proceedings of 1998 Computer-Supported Cooperative Work (CSCW 1998), Towards Adaptive workflow Systems Workshop*. Seattle, WA, USA, 1998

Heinl u. a. 1999
Heinl, Petra ; Horn, Stefan ; Jablonski, Stefan ; Neeb, Jens ; Stein, Katrin ; Teschke, Michael: A comprehensive approach to flexibility in workflow management systems. In: *WACC '99: Proceedings of the international joint conference on Work activities coordination and collaboration*. New York, NY, USA : ACM Press, 1999, S. 79–88

Hohpe und Woolf 2003
Hohpe, Gregor ; Woolf, Bobby: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2003

Houspanossian und Cilia 2005
Houspanossian, Alejandro ; Cilia, Mariano: Extending a BPEL Engine with AOP. In: *In Proceedings of the Argentinean Symposium on Software Engineering (ASSE'05)*. Rosario, Argentina, August 2005

Huang u. a. 2006
Huang, Yi ; Slominski, Aleksander ; Herath, Chathura ; Gannon, Dennis: WS-Messenger: A Web Services-Based Messaging System for Service-Oriented Grid Computing. In: *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006), 16-19 May 2006, Singapore*, IEEE Computer Society, 2006, S. 166–173. – URL http://www.extreme.indiana.edu/xgws/messenger/doc/HuangY-WSMessenger.pdf

Jablonski u. a. 1995
Jablonski, Stefan ; Böhm, Markus ; Schulze, Wolfgang: *Workflow-Management – Entwicklung von Anwendungen und Systemen*. dpunkt Verlag, Heidelberg, 1995

Johnson 2002
Johnson, Rod: *Expert One-on-One J2EE Design and Development (Programmer to Programmer)*. Wrox, 2002

Karastoyanova u. a. 2005a
Karastoyanova, Dimka ; Houspanossian, Alejandro ; Cilia, Mariano ; Leymann, Frank ; Buchmann, Alejandro: Extending BPEL for Run Time Adaptability. In: *Proceedings of the 9th International Enterprise Distributed Object Computing Conference (EDOC 2005), Enschede, The Netherlands*, September 2005

Karastoyanova u. a. 2005b
Karastoyanova, Dimka ; Leymann, Frank ; Buchmann, Alejandro P.: An Approach to Parameterizing Web Service Flows. In: Benatallah, Boualem (Hrsg.) ; Casati, Fabio (Hrsg.) ; Traverso, Paolo (Hrsg.): *Service-Oriented Computing - ICSOC 2005*, Springer-Verlag, Berlin, Heidelberg, Dezember 2005 (Lecture Notes in Computer Science Bd. 3826), S. 553 – 538

Karastoyanova u. a. 2006
Karastoyanova, Dimka ; Leymann, Frank ; Nitzsche, Jörg ; Wetzstein, Branimir ; Wutke, Daniel: Parameterized BPEL Processes: Concepts and Implementations. In: *Proceedings of the Internationall Conference on Business Process Management, BPM 2006, Vienna, Austria*, September 2006

Karastoyanova u. a. 2007
Karastoyanova, Dimka ; Leymann, Frank ; Schroth, Ralf ; Ortiz, Guadalupe: *BPEL'n' Aspects: Chiming Into Orchestrations*. 2007. – to appear

Kaye 2003
Kaye, Doug: *Loosely Coupled: The Missing Pieces of Web Services*. 1st Edition. Rds Associates Inc, 2003

Khalaf u. a. 2007
Khalaf, Rania ; Karastoyanova, Dimka ; Leymann, Frank ; Pottinger, Stefan ; Schroth, Ralf: *Controlling BPEL Execution*. 2007. – to appear

Kiczales u. a. 2001
Kiczales, Gregor ; Hilsdale, Erik ; Hugunin, Jim ; Kersten, Mik ; Palm, Jeffrey ; Griswold, William: Getting started with ASPECTJ. In: *Communications of the ACM* 44 (2001), Oktober, Nr. 10, S. 59–65

Kiczales u. a. 1997
Kiczales, Gregor ; Lamping, John ; Menhdhekar, Anurag ; Maeda, Chris ; Lopes, Cristina ; Loingtier, Jean-Marc ; Irwin, John: Aspect-Oriented Programming. In: Akşit, Mehmet (Hrsg.) ; Matsuoka, Satoshi (Hrsg.): *Proceedings European Conference on Object-Oriented Programming*. Berlin, Heidelberg, and New York : Springer-Verlag, 1997 (Lecture Notes in Computer Science Bd. 1241), S. 220–242

Kloppmann u. a. 2005
Kloppmann, Matthias ; Koenig, Dieter ; Leymann, Frank ; Pfau, Gerhard ; Rickayzen, Alan ; von Riegen, Claus ; Schmidt, Patrick ; Trickovic, Ivana: WS-BPEL Extension for People – BPEL4People / IBM Corp. und SAP AG. URL ftp://www6.software.ibm.com/software/developer/library/ws-bpel4people.pdf, Juli 2005. – Whitepaper

Le Hors u. a. 2004
Le Hors, Arnaud ; Le Hégaret, Philippe ; Wood, Lauren ; Nicol, Gavin ; Robie, Jonathan ; Champion, Mike ; Byrne, Steve: Document Object Model (DOM) Level 3 Core Specification / W3C. URL http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/, 2004. – Spezifikation

Leymann 2001
Leymann, Frank: Web Services Flow Language (WSFL 1.0) / IBM Corp. Somers, NY, USA, Mai 2001. – Spezifikation. – URL http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Leymann 2003
Leymann, Frank: Web Services: Distributed Applications Without Limits. In: Weikum, Gerhard (Hrsg.) ; Schöning, Harald (Hrsg.) ; Rahm, Erhard (Hrsg.): *BTW 2003, Datenbanksysteme für Business, Technologie und Web, Tagungsband der 10. BTW-Konferenz, 26.-28. Februar 2003, Leipzig*, GI, 2003 (Lecture Notes in Informatics Bd. 26), S. 2–23

Leymann und Roller 1997
Leymann, Frank ; Roller, Dieter: Workflow-Based Applications. In: *IBM Systems Journal* 36 (1997), Nr. 1, S. 102–123

Leymann und Roller 2000
Leymann, Frank ; Roller, Dieter: *Production Workflow - Concepts and Techniques*. Prentice Hall PTR, 2000

MacKenzie u. a. 2006
MacKenzie, C. M. ; Laskey, Ken ; McCabe, Francis ; Brown, Peter F. ; Metz, Rebekah: Reference Model for Service Oriented Architecture 1.0 / OASIS. URL http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf, August 2006. – Committee Specification

Martin 2002
Martin, Robert C.: *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002

Murphy und Schwanninger 2006
Murphy, Gail ; Schwanninger, Christa: Aspect-Oriented Programming. In: *IEEE Software* 23 (2006), Januar, Nr. 1, S. 20–23

Natis 2003
Natis, Yefrim V.: Service-Oriented Architecture Scenario / Gartner Inc. Stamford, CT, USA, April 2003 (AV-19-6751). – Whitepaper

Niblett und Graham 2005
Niblett, Peter ; Graham, Steve: Events and service-oriented architecture: The OASIS Web Service Notification specifications. In: *IBM Systems Journal* Bd. 44. IBM Corp., 2005, S. 869–886. – URL http://www.research.ibm.com/journal/sj/444/niblett.pdf

Nitzsche 2006
Nitzsche, Jörg: *Entwicklung eines Monitoring-Tools zur Unterstützung von parametrisierten Web Service Flows*, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, Diplomarbeit, März 2006. – 115 S. – URL http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-2388

Ortiz und Leymann 2006
Ortiz, Guadalupe ; Leymann, Frank: Combining WS-Policy and Aspect-Oriented Programming. In: *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW 2006)* IEEE (Veranst.), Februar 2006, S. 143 – 148

Ouyang u. a. 2005
Ouyang, Chun ; Verbeek, Eric ; Aalst, Wil M. van der ; Breutel, Stephen ; Dumas, Marlon ; Hofstede, Arthur H. ter: Formal Semantics and Analysis of Control Flow in WS-BPEL (Revised Version) / BPMCenter.org. URL http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2005/BPM-05-15.pdf, 2005 (BPM Center Report BPM-05-13). – Report

Paluszek 2007
Paluszek, Michael: *Coordinating Distributed Loops and Fault Handling, Transactional Scopes using WS-Coordination protocols layered on WS-BPEL services*, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, Diplomarbeit, 2007. – noch nicht erschienen

Parnas 1972
Parnas, David L.: On the criteria to be used in decomposing systems into modules. In: *Communications of the ACM* 15 (1972), Dezember, Nr. 12, S. 1053–1058

Popovici u. a. 2002
Popovici, Andrei ; Gross, Thomas ; Alonso, Gustavo: Dynamic weaving for aspect-oriented programming. In: *AOSD '02: Proceedings of the 1st international conference on Aspect-oriented software development*. New York, NY, USA : ACM Press, 2002, S. 141–147

Reichert und Dadam 1998
Reichert, Manfred ; Dadam, Peter: $ADEPT_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. In: *Journal of Intelligent Information Systems* 10 (1998), Nr. 2, S. 93–129

Reichert u. a. 2005
Reichert, Manfred ; Rinderle, Stefanie ; Kreher, Ulrich ; Dadam, Peter: Adaptive Process Management with ADEPT2. In: *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)* IEEE (Veranst.), 2005, S. 1113 – 1114

Rosenberg und Dustdar 2005
Rosenberg, Florian ; Dustdar, Schahram: Business Rules Integration in BPEL - A Service-Oriented Approach. In: *Proceedings of the Seventh IEEE International Conference on E-Commerce Technology (CEC 2005), 19-22 July 2005, München, Germany*, IEEE Computer Society, 2005, S. 476–479

Sato u. a. 2003
Sato, Yoshiki ; Chiba, Shigeru ; Tatsubori, Michiaki: A selective, just-in-time aspect weaver. In: *GPCE '03: Proceedings of the second international conference on Generative programming and component engineering*. New York, NY, USA : Springer-Verlag New York, Inc., 2003, S. 189–208

Schmidt und Assmann 1998
Schmidt, Rainer ; Assmann, Uwe: Extending Aspect-Oriented Programming in order to Flexibly Support Workflows. In: Lopes, Cristina V. (Hrsg.) ; Kiczales, Gregor (Hrsg.) ; Murphy, Gail (Hrsg.): *Int'l Workshop on Aspect Oriented Programming (ICSE 1998)*. Kyoto, Japan, April 1998

Sun Microsystems 2002
Hapner, Mark ; Burridge, Rich ; Sharma, Rahul ; Fialli, Joseph ; Stout, Kate: Java Message Service, Version 1.1 / Sun Microsystems. April 2002. – Specification

Sun Microsystems 2003
Sun Microsystems: *Java 2 PlatformEnterprise Edition Specification, Version 1.4*. November 2003. – URL http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf

Szyperski 1998
Szyperski, Clemens: *Component Software: Beyond Object-Oriented Programming*. ACM Press/Addison-Wesley Publishing Co., 1998

Thatte 2001
Thatte, Satish: XLANG – Web Services for Business Process Design / Microsoft Corp. URL http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, 2001. – Initial Public Draft

van der Aalst und Jablonski 2000
van der Aalst, Wil M. P. ; Jablonski, Stefan: Dealing with workflow change: identification of issues and solutions. In: *International Journal of Computer Systems Science and Engineering* 15 (2000), September, Nr. 5, S. 267–276

Vinoski 2004a
Vinoski, Steve: More Web Services Notifications. In: *IEEE Internet Computing* 08 (2004), Nr. 3, S. 90–93

Vinoski 2004b
Vinoski, Steve: Web Services Notifications. In: *IEEE Internet Computing* 08 (2004), Nr. 2, S. 86–90

Weerawarana u. a. 2005
Weerawarana, Sanjiva ; Curbera, Francisco ; Leymann, Frank ; Storey, Tony ; Ferguson, Donald F.: *Web Services Platform Architecture*. Prentice Hall, 2005

Wetzstein 2006

Wetzstein, Branimir: *Dynamische semantikbasierte Suche nach Web Services in parametrisierten BPEL-Prozessen*, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, Diplomarbeit, Januar 2006. – 90 S. – URL http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-2372

Workflow Management Coalition 1999

Workflow Management Coalition: Terminology & Glossary / Workflow Management Coalition. URL http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf, Februar 1999 (WFMC-TC-1011). – Whitepaper

Wutke 2006

Wutke, Daniel: *Erweiterung einer Workflow-Engine zur Unterstützung von parametrisierten Web Service Flows*, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, Diplomarbeit, März 2006. – 104 S. – URL http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-2401

Die Verfügbarkeit aller Links wurde am 18. November 2006 geprüft.

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen, als die angegebenen Hilfsmittel verwendet habe.

Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken nach bestem Wissen und Gewissen als solche kenntlich gemacht habe.

Stuttgart, den 21. November 2006 _____

Ralf Schroth