Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 2629

Integration und Nutzung kontextbezogener Nexus-Dienste in BPEL-Workflows

Peter Kaczmarczyk

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl. Inform. Matthias Wieland
begonnen am:	15. Januar 2007
beendet am:	17. Juli 2007
CR-Klassifikation:	C.2.4, D.2.12, H.3.3, H.4.1

Zusammenfassung

Die Entwicklung von kontextbasierten Anwendungen ist sehr komplex. Darum soll in dieser Arbeit untersucht werden, ob es möglich ist, durch Einsatz von Workflowsystemen kontextbasierte Anwendungen auf einfachere Weise zu entwickeln. Für die Bereitstellung von Kontextinformationen soll dazu die Nexus Plattform aus dem Sonderforschungsbereich 627 [Nexus] verwendet werden und für die Beschreibung von Workflows die BPEL Spezifikation [BPEL]. Die Integration der Kontextinformationen soll anhand von Beispielanwendungen evaluiert werden.

Danksagung

An erster Stelle möchte ich mich bei Prof. Dr. Frank Leymann für die interessante Themenstellung bedanken.

Mein besonderer Dank gilt Matthias Wieland, der mir während der Bearbeitung des Themas jederzeit mit Anregungen und konstruktiven Ratschlägen behilflich war.

Weiterhin möchte ich mich bei den Mitarbeitern des Nexus Projekts für ihre bereitwillige Hilfe bedanken, mit der auch Sie zum Gelingen dieser Arbeit beigetragen haben.

Schließlich möchte ich mich ganz herzlich bei meiner Familie, bei Anna Maria Ciba und bei meinem Freundeskreis für die Unterstützung während meines gesamten Studiums bedanken.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Aufgabenstellung 2
1.2 Aufbau der Arbeit 2

2 Grundlagen **5**
2.1 Nexus - Offene Infrastruktur für kontextsensitive Anwendungen 5
2.1.1 Nexus Architektur 7
2.1.2 Augmented World 8
2.1.3 Kontextserver 10
2.1.4 Föderation 11
2.1.5 Ereignisdienst 12
2.1.6 Value Added Services 13
2.2 XML und XSL 14
2.2.1 XML 14
2.2.2 XSLT 17
2.2.3 XPath 19
2.3 Serviceorientierte Architektur mit Web Services 20
2.3.1 SOAP 21
2.3.2 WSDL 23
2.4 Workflows 25
2.4.1 Klassifikation von Workflows 26
2.4.2 BPEL 27

3 Verwandte Arbeiten **31**
3.1 Ausgewählte Beispiele traditioneller kontextsensitiver Systeme 31
3.1.1 Context Toolkit 31
3.1.2 EasyMeeting 32
3.1.3 SOCAM 33
3.2 Ausgewählte Beispiele prozess- und serviceorientierter Kontextsysteme 34
3.2.1 ContextKernel 34
3.2.2 FollowMe 35
3.2.3 Service Globe 36
3.2.4 CAWE 37

4 Nexus in Workflows - Nutzung von Kontextdaten in BPEL Prozessen **39**
4.1 Analyse und Anforderungen 39
4.1.1 Datentypen, Datenaustauschformate und Funktionalität in Nexus 39
4.1.2 Transportprotokolle und Verteilung in Nexus 41
4.1.3 Funktionale und nichtfunktionale Anforderungen 43
4.2 Grobentwurf - Konzept und Architektur 44

4.2.1 Basiskonzept: Transformation und hierarchische *Integrationsprozesse* . 44
4.2.2 Entwicklung einer serviceorientierten Architektur für die Integration von Nexusdiensten . . . 46
4.2.3 Asynchrone Ereignisverarbeitung mit WS-Addressing . . . 47
4.3 Feinentwurf - Vorstellung der Integrationsprozesse . . . 48
4.3.1 ContextQuery . . . 49
4.3.2 ContextManipulation . . . 55
4.3.3 ContextModeling . . . 59
4.3.4 ContextEvent . . . 61
4.3.5 GetMap . . . 67
4.4 Implementierung . . . 70
4.4.1 ContextModeling, ContextQuery und ContextManipulation . . . 71
4.4.2 ContextEvent . . . 73
4.4.3 GetMap . . . 77
4.5 Inbetriebnahme . . . 78
4.6 Beispielprozess „Paketdienst” . . . 79
4.6.1 Modellierung von Boten und Lieferungen . . . 80
4.6.2 Workflow für den Lieferdienst - *ParcelService* . . . 80
4.6.3 Simulation der Bewegung von Boten und Lieferungen . . . 84

5 Bewertung der Ergebnisse 87
5.1 Auswertung der Anforderungen . . . 87
5.2 Einschränkungen . . . 88
5.3 Mögliche Erweiterungen . . . 89
5.3.1 Definition zusätzlicher Schnittstellen . . . 89
5.3.2 Bereitstellung von Diensten . . . 90
5.3.3 Definition von Kontextaktivitäten in BPEL . . . 90
5.3.4 Erweiterung der Architektur . . . 91
5.3.5 Enterprise Service Bus als Verbindungsglied . . . 91
5.4 Richtlinien für die Erstellung von Prozessen . . . 92

6 Fazit 95

Literaturverzeichnis 97

A Datentypen und Datenaustauschformate in Nexus 103
A.1 NSAT, NSAS, NSCS und erweiterte Schemata . . . 103
A.2 WKT und GML . . . 105
A.3 Nexus Locators . . . 105
A.4 AWQL, AWML und CRL . . . 106
A.5 Referenzierung von Schemaelementen . . . 110
A.6 ERL, ENL und EventInfo . . . 111
A.7 MapPL und MapML . . . 115

B WSDL Beschreibungen 119

C BPEL Prozesse 125

D XSLT Stylesheets 127

Abbildungsverzeichnis

2.1 Nexus Plattform und Komponenten [BDG+04] 8
2.2 Augmented World in Beziehung zur realen Welt [Meß01] 9
2.3 Modellabstraktionen Augmented World [Nic03] 10
2.4 Augmented Areas (SpaSe=**Spa**tial Model **Se**rver) [Sch03] 11
2.5 Föderation in der Nexus Plattform [Sch03] 12
2.6 Event Service Architektur [Bau03] 13
2.7 XML Schema Datentypen [XMLSchema] 17
2.8 Funktionsweise eines XSLT Prozessors [HZ03] 18
2.9 Baumstruktur für Element *mitarbeiter* 20
2.10 SOA Dreieck [WCL+05] 21
2.11 Web Service Architektur [WCL+05] 22
2.12 Aufbau einer SOAP Nachricht [Mel07] 23
2.13 Aufbau einer WSDL Beschreibung [WCL+05] 24
2.14 Apache Axis Architektur [WBFT04] 25
2.15 Klassifikation von Workflowsystemen [JMG02] 26

3.1 Beispielarchitektur einer Anwendung in Context Toolkit [SDA99] 32
3.2 SOCAM Architektur [GPZ05] 33
3.3 Context Kernel Architektur [ANGP03] 34
3.4 FollowMe Architektur [CBL+06] 35
3.5 Komponenten bei der Kontextverarbeitung in Service Globe [KK04] 36

4.1 Momentane Schnittstellen der Nexus Dienste 42
4.2 Verarbeitung komplexer Nachrichten in der Föderation 44
4.3 Integration von Nexusdiensten durch Transformation von Nachrichtenformaten 45
4.4 *Integrationsprozesse* zur Bereitstellung von Kontextdaten in Workflows 46
4.5 Architektur für die Integration von Nexusdiensten in Workflows 47
4.6 Aufruf eines asynchronen Nexusdiensts in einem Workflow 48
4.7 Prozessmodell von *ContextQuery* 49
4.8 Prozessmodell von *ContextQueryExclude* 52
4.9 Transformation der Dokumente bei *ContextQueryExclude* 53
4.10 Prozessmodell von *ContextManipulation* 57
4.11 Prozessmodell von *ContextModeling* 59
4.12 Prozessmodell von *ContextEvent* 63
4.13 Prozessmodell von *ContextEventOnEnterArea* 66
4.14 Transformation von ContextEventOnEnterAreaRequest nach ERL 67
4.15 Prozessmodell von *GetMap* 68
4.16 Sequenzdiagramm *Nachrichtenverarbeitung im SeifeServlet* (vereinfacht) . . . 71
4.17 Sequenzdiagramm *Ereignisregistrierung über WS-Schnittstelle* 74
4.18 Sequenzdiagramm *Ereignisregistrierung im RegistrationHandler* 75

4.19 Sequenzdiagramm *Asynchrone Ereignisbenachrichtigung* 76
4.20 Testumgebung der Implementierung 79
4.21 Workflowmodell Lieferdienst 81
4.22 Workflowmodell PickupParcel 82
4.23 Workflowmodell DeliverParcel 84
4.24 Anwendungsoberfläche zum Simulieren der Objektbewegung 85

5.1 Nexusdienste in einem ESB 91

A.1 Übersicht Nexus Bezeichner [BDG+04] 105
A.2 Zugriff auf Dokumentteile mit der Punktnotation 111

Abkürzungsverzeichnis

Abb.	Abbildung
API	Application Programming Interface
ASR	Area Service Register
AWM	Augmented World Model
AWML	Augmented World Modeling Language
AWQL	Augmented World Query Language
AWS	Augmented World Schema
BPEL	Business Process Execution Language
bzw.	beziehungsweise
CORBA	Common Object Request Broker Architecture
CRL	Change Report Language
EAS	Extended Attribute Schema
ECS	Extended Class Schema
ENL	Event Notification Language
ERL	Event Registration Language
GML	Geography Markup Language
HTML	HyperText Markup Language
IT	Informationstechnologie
MapML	Map Modeling Language
MapPL	Map Predicate Language
NSAS	Nexus Standard Attribute Schema
NSAT	Nexus Standard Attribute Types Schema
NSAT	Nexus Standard Attribute Types
NSCS	Nexus Standard Class Schema
o.g.	oben genannten
QoS	Quality of Service
RPC	Remote Procedure Call
s., s.o., s.u.	siehe, siehe oben, siehe unten
SOA	Serviceoriented Architecture
u.a.	unter anderem
UDDI	Universal Description and Discovery Interface
VIT	Virtual Information Tower
WKT	Well-known Text Format
WS	Web Service
WSDL	Web Service Description Language
WSFL	Web Services Flow Language
XML	Extensible Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations
z.B.	zum Beispiel

1 Einleitung

In der Industrie wird ständig nach neuen Methoden und Vorgehensweisen zur Verbesserung der Produktqualität und zur Einsparung von Kosten in der Produktion gesucht. Der Fertigungsprozess ist dabei, von der Idee bis zur Marktreife des Produkts, einer ständigen Entwicklung ausgesetzt. Während dieser Entwicklung fließt das Wissen, die Fertigkeit und die Erfahrung vieler am Prozess beteiligter Menschen mit in den Prozess ein. Die Prozesse, die ein Unternehmen zur Fertigung seiner Produkte besitzt, bestimmen den wesentlichen Erfolg eines Unternehmens. Sie sind wesentlich beteiligt an der Wertbildung eines Unternehmens [LR00]. Die einzelnen Prozesse eines Unternehmens werden auch als *Geschäftsprozesse* bezeichnet. Ein Geschäftsprozess ist beispielsweise für einen Automobilhersteller auf oberster Ebene die Produktion eines Fahrzeugs und ein untergeordneter Teilprozess darin ist die Fertigung des Motors. Die Fertigung eines Motors besteht dabei wiederum aus einzelnen Teilschritten, die durchgeführt werden müssen, z.B. der Beschaffung der Motorenteile, die Montage der Teile oder die Prüfung der montierten Motoren. Es bestehen bereits viele Ansätze, diese Prozesse mit informationstechnischen Mitteln zu unterstützen, z.B. durch Produktplanungssysteme, mit denen die Fertigung eines Produkts durch die Konstrukteure im Büro geplant und gesteuert werden kann.
Zusätzlich zu dieser Entwicklung erwächst in den letzten Jahren das Bedürfnis nach einer neuen Generation von Fabrikationsanlagen, die ausgezeichnet sind durch „intelligente" Prozesse. Diese Art der Fabrikationsanlage wird auch als *smart factory* [BJRW03] bezeichnet. Der Begriff *smart factory* bzw. *smart environment* hat seinen Ursprung in den Bereichen der *context awareness* [Ubiquitous] und des *pervasive computing* [Pervasive]. Dabei geht es um die Vision, dass in Zukunft die Umgebung eines Menschen durchdrungen ist von einer Vielzahl von Computersystemen, z.B. im Haushalt oder in Fahrzeugen, und diese Systeme sich ihrer Umgebung bewusst sind, z.B. die aktuelle Raumtemperatur oder die Anzahl der Insassen kennen. Aufgrund der Wahrnehmung dieser *Kontextinformationen* ergeben sich zusätzliche Vorteile für eine *smart factory*. Die Vorteile finden sich u.a. im Betriebsmittelmanagement [BJRW03]. In einer intelligenten Fabrik sind sich dabei die einzelnen Produktionsprozesse der vorhandenen Betriebsmittel bewusst und können sich beispielsweise bei einer Betriebsmittelknappheit automatisch anpassen oder umstellen. Zusätzliche Vorteile ergeben sich bei der Wartung und Instandhaltung von Maschinen. In einer intelligenten Fabrik sind sich die Maschinen ihres Zustandes bewusst und können so im Falle einer Fehlfunktion einen Wartungsprozess initiieren [WLJ+06]. Weitere sinnvolle Szenarien in einer *smart factory*, die zu einer Verbesserung des Prozessablaufs führen, sind denkbar.
Zum Ende des 20. Jahrhunderts wird für die informationstechnische Unterstützung von Geschäftsprozessen die *Workflow* Technologie vorgestellt [LR00]. Workflowanwendungen haben gegenüber den klassischen Anwendungen den wesentlichen Vorteil, dass sie entlang der Geschäftsprozesse eines Unternehmens entwickelt werden. Diesen Anspruch erfüllen sie, indem sie anhand von Prozessbeschreibungen realisiert werden. Schritte im Geschäftsprozess finden sich exakt in der Workflowanwendung wieder. Ändert sich ein Geschäftsprozess so kann in einer Workflowanwendung der entsprechende Prozessteil auf einfache Weise identifiziert und anschließend geändert werden. Dadurch können Verbesserungen in Geschäftsprozessen auf effektive Weise in Workflowanwendungen aufgegriffen werden. Auf der anderen Seite können auch bei der Model-

lierung von Workflowanwendungen Ideen zur Verbesserung eines Geschäftsprozesses entstehen. Das gemeinsame Verbindungsglied - *der Prozess* - fördert somit eine Wechselwirkung. Darüber hinaus bieten Workflowanwendungen einen hohen Grad an Automatisierung für Prozessteile.

1.1 Aufgabenstellung

Die Vorteile der Workflow Technologie sollen mit der Vision der *smart factory* verknüpft werden. Zu diesem Zweck sollen in dieser Arbeit Kontextinformationen in Workflowumgebungen verfügbar gemacht werden. Das Ziel dieser Arbeit ist es, die Grundlage für kontextsensitive Workflowanwendungen zu schaffen, damit anschließend eine Abbildung von Teilen eines intelligenten Prozesses aus der *smart factory* in Form eines Workflows erfolgen kann. Für die Bereitstellung der Kontextinformationen wird in dieser Arbeit die *Nexus Plattform* betrachtet, die im Rahmen des Sonderforschungsbereichs 627 [Nexus] entstanden ist. Von der Nexus Plattform werden zahlreiche Dienste für die Entwicklung kontextsensitiver Anwendungen zur Verfügung gestellt. Für die Beschreibung von Workflows wird die BPEL Spezifikation [BPEL] verwendet. Um die in BPEL beschriebenen Prozesse für die *smart factory* intelligent zu machen, ist folglich eine Verbindung der BPEL Prozesse mit den auf der Nexus Plattform zur Verfügung gestellten Kontextinformationen erforderlich.

1.2 Aufbau der Arbeit

Zur Klärung der Frage, wie eine Integration von Kontextinformationen in Workflowanwendungen erfolgen kann, werden in Kapitel 2 zuerst *grundlegende Kenntnisse* vermittelt, die für ein besseres Verständnis der Thematik Voraussetzung sind. Dabei wird zum einen die Nexus Plattform mit ihrer Funktionalität beschrieben. Zum anderen werden Workflows näher betrachtet und die Möglichkeit, diese durch BPEL zu beschreiben. Für beide Bereiche sind zudem Kenntnisse der XML Spezifikation [XML] erforderlich. Schließlich wird noch die Web Service Architektur behandelt, die für die Bereitstellung von Diensten in BPEL Prozessen nützlich ist.
In Kapitel 3 werden anschließend *verwandte Arbeiten* und deren Ansätze für die Bereitstellung von Kontextinformationen in Anwendungen vorgestellt. Dabei wird eine Gegenüberstellung dieser Systeme mit der Nexus Plattform gemacht.
Kapitel 4 widmet sich schließlich dem Hauptthema dieser Arbeit - der *Anbindung der Workflowanwendungen an die Nexus Plattform*. Das Kapitel gliedert sich in die üblichen Phasen, die bei einem Softwareprojekt durchgeführt werden. In der Analyse wird eine Momentaufnahme der aktuellen Situation gemacht, es wird die Problemstellung erarbeitet und es werden Anforderungen an das neue System formuliert. Im Grobentwurf wird eine Lösung auf hoher Ebene für die Anbindung der Nexus Plattform vorgestellt, die dann im Feinentwurf ausgearbeitet wird. Dabei wird im Feinentwurf das Konzept der *Integrationsprozesse* zur Anbindung einer Workflowanwendung an die Nexus Plattform ausgearbeitet. In der Implementierung erfolgt anschließend die Realisierung der feineren Entwurfseinheiten, z.B. durch die Programmierung der Web Service Schnittstellen. Die Inbetriebnahme zeigt die Installation der Komponenten auf einer Testplattform und eine abschließende Beispielanwendung soll die Verwendung der erarbeiteten Lösung abrunden.
In Kapitel 5 erfolgt eine abschließende *Bewertung der Ergebnisse* aus dem Hauptteil. Es wird die Umsetzung der Anforderungen, die für das neue System formuliert wurden, betrachtet. Zusätzlich werden die Einschränkungen des hier gewählten Ansatzes aufgeführt. In einem

weiteren Abschnitt werden mögliche Erweiterungen diskutiert, die eine Fortführung der Arbeit ermöglichen. Schließlich werden noch Richtlinien für die Erstellung von kontextsensitiven Workflowanwendungen vorgestellt.

2 Grundlagen

In diesem Kapitel werden grundlegende Kenntnisse vermittelt, die für ein besseres Verständnis der Arbeit erforderlich sind. Dafür wird zuerst der Begriff des *Kontexts* geklärt, sowie die Notwendigkeit und die Eigenschaften *kontextsensitiver Systeme* aufgezeigt. Anschließend wird die *Nexus Plattform* vorgestellt, die diese Forderungen an ein kontextsensitives System realisiert und später im Hauptteil dieser Arbeit für die Bereitstellung von Kontextinformationen in Workflowumgebungen genutzt werden soll.
In vielen Systemen spielt die *Extensible Markup Language* [XML] zur Beschreibung und zum Austausch von Daten eine immer größer werdende Rolle. Deshalb sollen in Abschnitt 2.2 grundlegende Kenntnisse der XML Spezifikation vermittelt werden. Hier wird auch die *Transformation von XML* Dokumenten [XSLT] behandelt, die für eine spätere Integration der Nexusdienste in Workflows zur Umwandlung von Nachrichtenformaten genutzt wird.
Für Workflowumgebungen ist es zudem vorteilhaft, wenn die Nexusdienste über eine Architektur bereitgestellt werden, die eine einfache Einbindung der Dienste ermöglicht. Zu diesem Zweck wird in Abschnitt 2.3 eine *serviceorientierte Architektur* vorgestellt, die Web Service Architektur, die für die Beschreibung und Bereitstellung von lose gekoppelten Diensten zahlreiche Vorteile bietet.
Für die informationstechnische Unterstützung von Geschäftsprozessen in Unternehmen werden zunehmend *Workflowsysteme* eingesetzt. Deshalb sollen im letzten Abschnitt 2.4 der Begriff des *Workflows* geklärt werden und eine Klassifikation von Workflowsystemen gegeben werden. Anschließend werden die wesentlichen Bestandteile der Business Process Execution Language [BPEL] vorgestellt, die später für die Beschreibung von Workflows eingesetzt wird.

2.1 Nexus - Offene Infrastruktur für kontextsensitive Anwendungen

Bevor die Beschreibung der Nexus Plattform erfolgt, müssen die wesentlichen Begriffe und Eigenschaften aus dem Umfeld kontextsensitiver Systeme geklärt werden. In diesem Absatz werden dazu die Begriffe *context*, *context awareness*, *ubiquitous computing* und *pervasive computing* betrachtet. In vielen Definitionen werden context und context awareness meist unscharf beschrieben, z.B. „Kontext ist der Ort und die Daten, die an einem Ort abgelegt sind".
In [ADB+99] wird deshalb eine bessere Definition von Kontext gegeben:

> „Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

Es gibt also die folgenden wesentlichen Bausteine beim Kontext:

- Die Situation, die einen Zustand von Entitäten festlegt.
- Eine Entität steht stellvertretend für Objekte, Plätze und Personen, die für die Situation

von Bedeutung sind.

- Kontextinformationen sind alle Informationen zu diesen Objekten in dieser Situation.

Darauf aufbauend wird in [ADB+99] *context awareness* erläutert:

> „A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user task.“

Eine Anwendung ist also *context aware*, falls sie Kontextinformationen verwendet (s. Definition *context*). Dabei verwendet sie nur die Kontextinformationen, die für die entsprechende Aufgabe, die der Benutzer ausführen möchte, sinnvoll sind. Nach [ADB+99] ist also eine Anwendung, die einen Benutzer durch die Sehenswürdigkeiten einer Stadt führen soll, nicht *context aware*, wenn sie dem Benutzer, anstatt der Sehenswürdigkeiten an einem bestimmten Ort, den Herzschlag oder die Gehgeschwindigkeit anzeigt.
Mark Weiser spricht diesbezüglich von einem Paradigmenwechsel bei der Verwendung von Rechnersystemen, der später unter dem Schlagwort *Ubiquitous Computing* bekannt werden soll [Ubiquitous]. Weiser unterscheidet dabei unterschiedliche Phasen der Evolution von Computersystemen. Zuerst war die Zeit der Mainframe Rechner, in der viele Personen an einem Rechner gearbeitet haben. Darauf folgte die Zeit des Personal Computers, in der beinahe jede Person einen eigenen Rechner besitzt, an dem sie ihre Arbeit erledigen kann. Weiser sagt für die Zukunft der Rechnersysteme eine Zeit voraus, in der eine Person von vielen Rechnern umgeben sein wird und die Dienste dieser Rechner nutzen wird.

> „... a new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselves to vanish into the background.“ [Ubiquitous]

In seiner Vision gibt es eine Vielzahl von Rechnern in der menschlichen Umgebung, die am Ende nahezu unsichtbar im Hintergrund Aufgaben erfüllen. Der Mensch erkennt diese Rechner als selbstverständlich an und nimmt deren Nutzen ebenso flüchtig wahr, wie z.B. den Nutzen von Straßenschildern, die ihm täglich den Weg zur Arbeit säumen.
Bei IBM wird bezüglich ähnlicher Entwicklungen von einer Metamorphose gesprochen:

> „... from a world of computers that are seen as such and used by humans as distinct machines, into a world of sophisticated, computerized, networked machines that are not seen as computers nor used as such by humans.“ [Pervasive]

Von IBM wird dafür der Begriff des *pervasive computing* geprägt, eine Umgebung die von Rechnern in allen möglichen Bereichen durchsetzt ist.
In [HIR01] wird schließlich auf die Notwendigkeit von *context awareness* bei *pervasive* bzw. *ubiquitous computing* hingewiesen:

> „In the future, the computing landscape will evolve into an environment in which computers are autonomous devices that provide largely invisible support for tasks performed by users...
> Applications will have greater awareness of context, and thus will be able to provide more intelligent services that reduce the burden on users to direct and interact with applications.“

Das bedeutet, dass für die Vision einer von Rechnern durchdrängten Umgebung, in der die Rechner vom Menschen als selbstverständlich, ja nahezu als „natürlich“ angesehen werden, die Rechner und Systeme eine Eigenschaft aufweisen müssen, die es ihnen erlaubt sich in den Hintergrund abzusetzen, nämlich die Eigenschaft der *context awareness*. Nur mit dem Wissen,

welches die Rechner aus ihrer Umgebung aufnehmen können, gelingt es ihnen, gleichzeitig autonom Aufgaben durchzuführen und sich der jeweiligen Umgebung anzupassen.
Im folgenden Absatz sollen die wesentlichen Eigenschaften aufgezählt werden, die sich aus den obigen Anforderungen an kontextsensitive Systeme ergeben. Diese werden bei [Sch05] folgendermaßen formuliert:

- Überwachung der Umgebung und Bereitstellung der Informationen
- Unterstützung von expliziter und impliziter Benutzerinteraktion
- Unterstützung der Mobilität physischer Objekte
- Modellierung relevanter Weltausschnitte
- Bereitstellen von Kommunikationsverbindungen
- Mechanismen zur effizienten Verwaltung von Energie
- Integration mit bisheriger Hard- und Software sowie klassischen Middleware-Systemen
- Benachrichtigung über Objektzustände

Als klassische Middleware werden in [Sch05] Systeme bezeichnet, die auf RPC- oder CORBA-Konzepten basieren. Als aktuelle Entwicklungen werden Web Service- oder Peer-to-Peer-Lösungen angesehen (siehe dazu auch [Ben04]). Weitere allgemeine Anforderungen, die dort an eine Middleware für verteilte Systeme gestellt werden, sind:

- Gemeinsame Nutzung von Ressourcen
- Offenheit, um neue Einheiten hinzufügen zu können
- Nebenläufigkeit, um Berechnungen effizienter zu gestalten
- Skalierbarkeit für wachsende Systeme
- Fehlertoleranz, um Ausfall des Gesamtsystems abzufangen
- Transparenz, um Komplexität zu verbergen

2.1.1 Nexus Architektur

In diesem und den folgenden Abschnitten werden die Bestandteile der Nexus Plattform näher beschrieben. Die Architektur von Nexus ist in drei verschiedene Schichten eingeteilt. Die Anwendungsschicht, die Föderationsschicht und die Serviceschicht. In Abbildung 2.1 sind die Plattform und deren Komponenten auf den drei verschiedenen Schichten abgebildet.
Auf der **Serviceschicht** (*service tier*) befinden sich die Kontextserver (*Context Server*). Diese (siehe Abschnitt 2.1.3) halten zu bestimmten Gebieten die Daten von Objekten bereit, die sich in diesen Gebieten befinden. Bei der Modellierung dieser Daten wird unterschieden zwischen beweglichen und festen Objekten, z.B. Fahrzeugen oder Gebäuden. Diese Informationen werden bestimmten Objekten und räumlichen Umgebungen der tatsächlich vom Menschen wahrgenommenen Welt zugewiesen, weshalb dieses „erweiterte" Weltbild auch als *Augmented World* bezeichnet wird. Für die Modellierung der Objekte in der Augmented World wird dazu in Nexus das Augmented World Modell eingeführt (siehe Abschnitt 2.1.2).
Die **Föderationsschicht** (*federation tier*), die das „Herzstück" der Nexus Plattform darstellt, ermöglicht kontextsensitiven Anwendungen der darüber liegenden Schicht den Zugriff auf die zahlreichen kontextbezogenen Funktionen von Nexus. Sie beherbergt die *Nexus Nodes* und das *Area Service Register* (ASR). Die Nexus Nodes sind Server, auf denen die unterschiedlichen

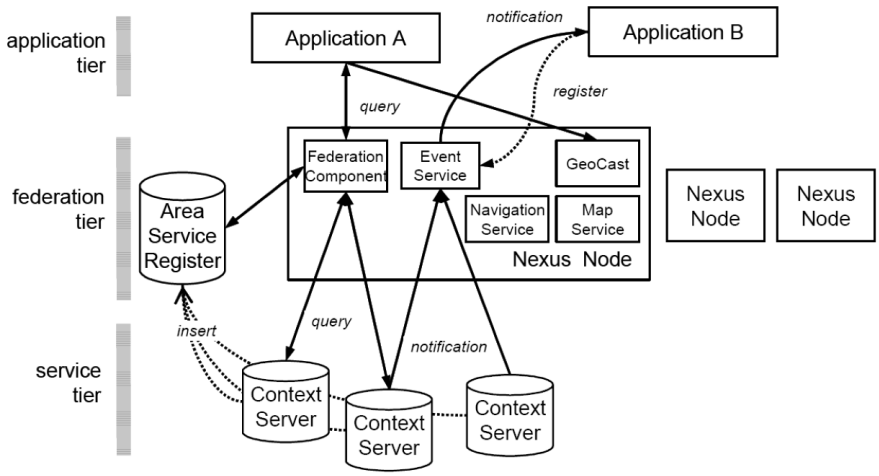

Abbildung 2.1: Nexus Plattform und Komponenten [BDG+04]

Dienste der Nexus Plattform gebündelt sind, u.a. die Föderation (*federation component*, siehe Abschnitt 2.1.4), die eine Abfrage von Daten ermöglicht, der Ereignisdienst (*event service*, siehe Abschnitt 2.1.5), der das Beobachten von Ereignissen erlaubt und weitere Mehrwertdienste (*value added services*, siehe Abschnitt 2.1.6) wie z.B. der Map Service, der die Erstellung von geografischen Karten vornimmt. Das ASR ist ein Verzeichnis, in dem sich die Kontextserver aus der Serviceschicht registrieren können. Die Anmeldung erfolgt unter Angabe der räumlichen Region (*augmented area*), die vom Kontextserver abgedeckt wird, der Angabe sämtlicher Objekttypen, die der Kontextserver kennt und der Adresse, unter der ein Kontextserver erreicht werden kann. Das ASR setzt somit mehrere geografische Gebiete (*augmented areas*) mit den dort vorkommenden Objekten in Beziehung und die Kontextserver halten die Daten für die Objekte aus einem dieser Gebiete bereit.
Die **Anwendungsschicht** (*application tier*) beinhaltet sämtliche Anwendungen, die die Dienste der unteren Schichten der Nexus Plattform benutzen, z.B. eine Anwendung zur Erkundung von Sehenswürdigkeiten, in der die Dienste der Föderation verwendet werden, um Kontextinformationen an einem bestimmten Ort abzurufen. Der Anwendungsschicht stehen dazu verschiedene Schnittstellen zur Verfügung (siehe Abschnitt 4.1).

2.1.2 Augmented World

Die Augmented World, wie sie von der Nexus Plattform zur Verfügung gestellt und von den Anwendungen wahrgenommen wird, wird beschrieben durch das *Augmented World Model* (AWM). In Abbildung 2.2 ist dargestellt, wie reale Objekte in Objekte der Augmented World abstrahiert werden. Die Vereinigung der mittleren und der unteren Ebene repräsentiert das AWM. Die unterste Ebene enthält dabei Informationen, die über das Internet zugreifbar sind und an „virtuellen Litfasssäulen" (*virtual information towers*, VIT) angeheftet werden, um Informationen, die im Internet verfügbar sind, dem Ort, an dem der VIT steht, zuzuweisen [Meß01]. Die Augmented World ist also eine Verschmelzung von realen Objekten mit virtuellen Objekten zu einer „neu wahrgenommenen" Wirklichkeit, basierend auf einem definierten Modell, das die Daten über diese „neu wahrgenommene Welt" enthält. Die Modellierung der Daten im

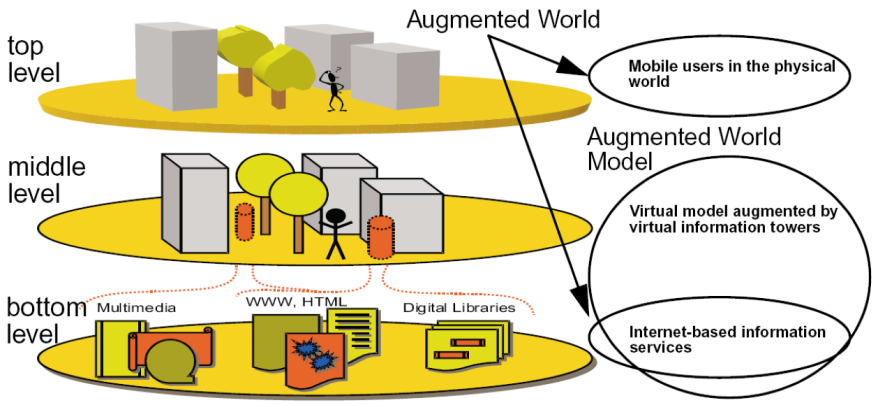

Abbildung 2.2: Augmented World in Beziehung zur realen Welt [Meß01]

Augmented World Model erfolgt dabei auf Basis einer Taxonomie von Datentypen. Dabei wirkt die Taxonomie bei der Modellierung neuer Daten wie eine Schablone, indem sie die wesentlichen Eigenschaften von Objekten für kontextsensitive Umgebungen sicherstellt. Diese Taxonomie wird auch als *Augmented World Schema* (AWS) bezeichnet. Abbildung 2.3 verdeutlicht die Beziehung der einzelnen Begriffe Augmented World, Augmented World Model und Augmented World Schema zueinander. Die Beschreibung der realen Objekte auf der untersten Ebene, wie z.B. für einen Sessel, wird auf der obersten Ebene durch ein objektorientiertes Metadatenmodell festgelegt, in dem Klassen und Attribute stellvertretend für die verschiedenen Arten von Objekten und Objekteigenschaften stehen. Auf Basis des Metadatenmodells kann ein Datenmodell erstellt werden, in dem festgelegt wird, was für Klassen und Attribute von Objekten wichtig sind. Es sollen beispielsweise Objekte der Klasse Sitzgelegenheit existieren und sie sollen das Attribut Farbe besitzen. Daten werden dann aufgrund diese Datenmodells erstellt, z.B. ein Sessel, der grün ist. Das Datenmodell aus dem Bild entspricht dem Augmented World Schema in Nexus. Das Schema setzt sich aus folgenden Bestandteilen zusammen (siehe auch Anhang A.1):

- Standarddatentypen (*Nexus Standard Attribute Types*, NSAT),
- Standardattribute (*Nexus Standard Attribute Schema*, NSAS) und
- Standardklassen (*Nexus Standard Class Schema*, NSCS).

Die Schemata werden dabei von den Kontextservern für die interne Repräsentation der Objekte verwendet. Wird ein Gebäude modelliert, so gibt es die Möglichkeit die Klasse *Building* aus dem NSCS als Klasse für das Objekt zu wählen, da diese Klasse bereits in den Standardklassen von Nexus vertreten ist. Um die Nexus Plattform offen für zukünftige Anwendungen zu halten, wird die Möglichkeit angeboten, das Augmented World Schema zu erweitern. Die erweiterten Schemata basieren dabei auf den Standardtypen, Standardattributen oder auf den Standardklassen. Wird ein erweitertes Klassenschema für Fabriken benötigt, so kann in einem erweiterten Schema z.B. die Lagerhalle als entsprechende Klasse definiert werden, wobei als Basisklasse dazu Building aus den Standardklassen verwendet wird. Erweiterte Klassenschemata bzw. Attributschemata werden dann als *Extended Class Schema* (ECS) und *Extended Attribute Schema* (EAS) bezeichnet.

Die Verwendung eines Augmented World Schema hat den Zweck, dem Modell bereits eine gewisse Semantik durch vordefinierte Klassen und Attribute zu geben. Durch optionale Attribute

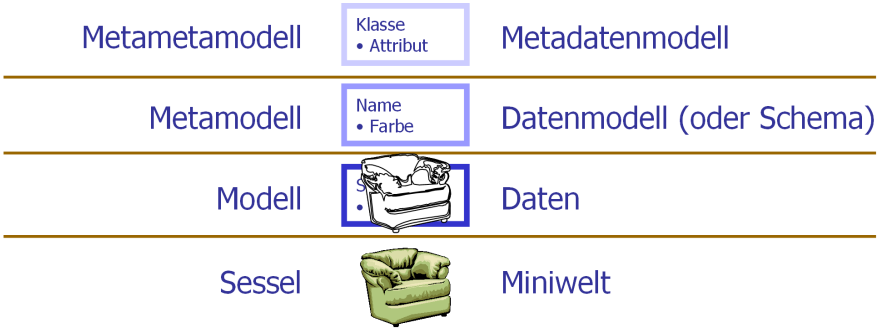

Abbildung 2.3: Modellabstraktionen Augmented World [Nic03]

wird zudem die Integration von Diensten erleichtert, da Attribute nicht unbedingt gesetzt werden müssen. Die Konzepte des Augmented World Model durchdringen alle Ebenen der Nexus Plattform und sind sowohl in den Diensten der Föderationsschicht als auch in den Anwendungen der Anwendungsschicht zu berücksichtigen. Deshalb muss das Augmented World Model auch in einer Workflowanwendung berücksichtigt werden, in welche die Nexusdienste integriert werden.

2.1.3 Kontextserver

Die Daten der Augmented World werden über Datenbanken in der Nexus Plattform zur Verfügung gestellt, die intern das Augmented World Schema verarbeiten können. Solche Datenbanken werden als Kontextserver bezeichnet, da sie Daten von Objekten in ihrer räumlichen Umgebung liefern. Die Nexus Plattform unterscheidet zwei Arten von Kontextservern:

- *Spatial Model Server* für statische Objekte und
- *Location Server* für mobile Objekte.

Ein Gebäude ist beispielsweise ein statisches Objekt, da sich dessen Position nicht ändert. Mobile Objekte hingegen wechseln oft ihre Position. Hierzu zählen z.B. Personen oder Fahrzeuge. Mobile Objekte wurden aufgrund der zusätzlichen Eigenschaft des ständigen Positionswechsels und der Übergabe von Objekten (*handover*) zwischen zwei Kontextservern bei einem Gebietsübergang durch einen eigenständigen Dienst, den *Location Service* [LR02], verwaltet. Der Location Service besteht dabei aus mehreren hierarchisch angeordneten *Location Servern*, die für die Verfolgung der Objekte zuständig sind (ähnlich der Verfolgung von Mobiltelefonen, bei denen ständig auf den Wechsel der Mobilfunkzellen reagiert werden muss). Der Fokus lag also auf der Bewegungsverfolgung von Objekten (siehe auch Abschnitt 5.2). Beim *Spatial Model Server* hingegen bewegen sich die Objekte nicht und es steht deshalb die Modellierung komplexer Objekteigenschaften im Vordergrund. Alle Spatial Model Server zusammen definieren die statische Seite des Augmented World Modells und alle Location Server zusammen definieren die dynamische Seite. Da ein Kontextserver als Kontextinformation auch die Position von Objekten speichert, erfolgt zusätzlich die Angabe des Gebiets, welches vom Kontextserver abgedeckt wird, der sog. *Augmented Area* (siehe Abb. 2.4). Die Augmented Area bezeichnet dabei ein logisches Gebiet, d.h. es können problemlos mehrere Augmented Areas für ein und dasselbe reale Gebiet existieren. Ein Kontextserver kann durchaus ein oder mehrere Augmented Areas abdecken. Eine Augmented Area sollte jedoch aus Effizienzgründen vollständig auf einem Kontextserver laufen. Zusätzlich darf aus Konsistenzgründen ein Objekt nur einmal in einer Augmented Area

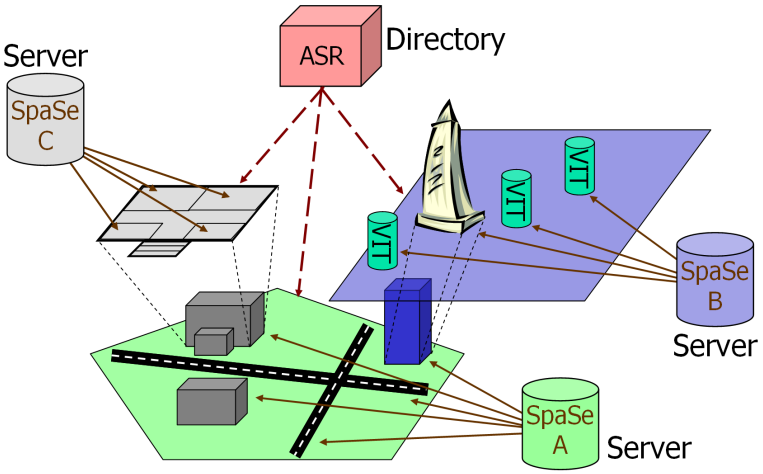

Abbildung 2.4: Augmented Areas (SpaSe=**Spa**tial Model **Se**rver) [Sch03]

existieren. Für jedes Objekt kann es jedoch ein oder mehrere Repräsentationen in verschiedenen Augmented Areas geben, die dann in einem späteren Schritt zusammengeführt werden müssen (siehe Abschnitt 2.1.4). Für Anwendungen und somit auch für zukünftige Workflows ist die Unterscheidung von statischen und mobilen Objekten transparent. Die Föderationsschicht liefert den Anwendungen grundsätzlich Objektdaten unabhängig von dieser Eigenschaft der Objekte. Die Anwendung kann jedoch auf Grundlage des Objekttyps jederzeit feststellen, ob es sich um ein statisches oder mobiles Objekt handelt, und die entsprechenden Eigenschaften dieser Objekte ausnützen (wie z.B. die Verfolgung der Position). Zusätzlich können Anwendungen durch die Definition von Augmented Areas gezielt auf die Daten einer Region zugreifen.

2.1.4 Föderation

Auf der Föderationsschicht liegen das ASR und die Nexus Nodes. Auf den Nexus Nodes befindet sich u.a. die Föderationskomponente. Im weiteren Verlauf der Arbeit wird für die Föderationskomponente der Begriff *Föderation* verwendet. Abbildung 2.5 zeigt die Föderation zwischen der Anwendungsschicht und der Serviceschicht. Die Föderation hat die besondere Aufgabe die unterschiedlichen Daten der verschiedenen Kontextserver zu einem Gesamtergebnis zu verschmelzen. Jeder Kontextserver, der sich an der Nexus Plattform beteiligt, liefert Daten zu den Objekten, die sich in seinem Gebiet befinden. Dabei können zu einem Objekt auf den verschiedenen Kontextservern unterschiedliche Informationen abgelegt sein. Angenommen in einer Fertigungshalle für Fahrzeuggetriebe befinden sich die Abteilungen „Montage" und „Prüfung". In der Montage wird das Getriebe zusammengesetzt und in der Prüfung werden ausgewählte Getriebe stichprobenartig getestet. Die Prüfungsabteilung stellt einen Kontextserver bereit, der prüfungsrelevante Daten zu einem Getriebe liefert und die Montageabteilung liefert Daten zum bisherigen Montageablauf des gleichen Getriebes. Erfolgt nun eine Anfrage von einer Anwendung nach einem bestimmten Fahrzeuggetriebe, so ist es möglich, dass sowohl montage- als auch prüfungsrelevante Daten für das Getriebe angefallen sind. Die Probleme, die bei der Verbindung dieser Informationen auftauchen, werden von der Föderation gelöst. Mit der Anfrage der Anwendung (*query*) wird die Föderation mit Hilfe des ASR die Kontextserver ermitteln, die Daten für das jeweilige Objekt zur Verfügung stellen. Die ausgewählten Kontextserver werden nun alle von der Föderation kontaktiert (*subqueries*), mit der Bitte, Daten über das Objekt zu senden. Jeder Kontextserver

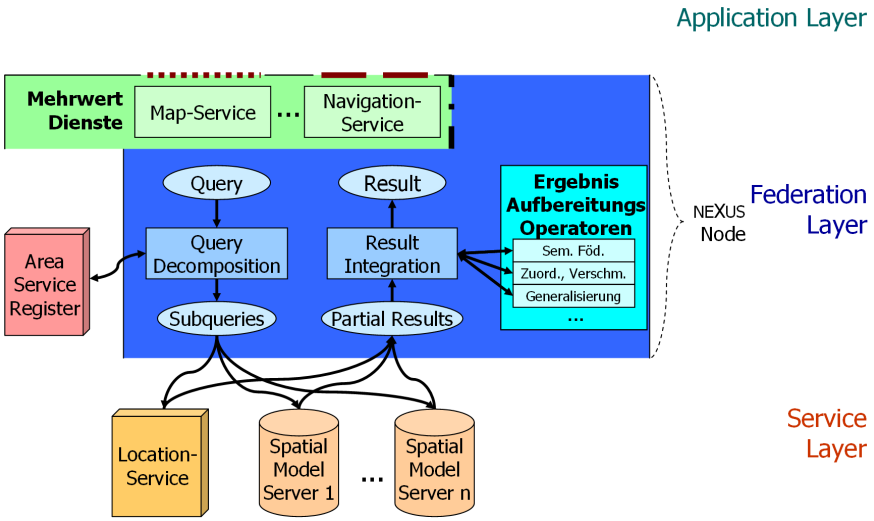

Abbildung 2.5: Föderation in der Nexus Plattform [Sch03]

sendet seine Daten an die Föderation (*partial results*). Die Föderation implementiert unterschiedliche Aufbereitungsfunktionen, mit denen sie die einzelnen Ergebnisse zu einem Gesamtergebnis verbindet. Eine mögliche Aufbereitungsfunktion wäre z.B. die einfache Aneinanderreihung der Einzelergebnisse zu einem Gesamtergebnis.
Anwendungen müssen die Aufbereitung der Daten bei ihrer Verarbeitung berücksichtigen. So könnte es beispielsweise bei der zuvor genannten Aufbereitungsfunktion im Gesamtergebnis zu Mehrfachnennungen von Attributen kommen. Außerdem könnte ein Objekt unterschiedliche Typen besitzen, z.B. wenn die Prüfungsabteilung für die Modellierung des Getriebes anstelle des Typs „Getriebe" den Typ „Prüfteil" verwendet. Detaillierte Informationen zur Föderationskomponente gibt es bei [Lia02].

2.1.5 Ereignisdienst

Der Ereignisdienst (*Event Service*) ermöglicht das Beobachten von Ereignissen auf der Nexus Plattform. Ein Ereignis steht dabei für eine Veränderung im Modellzustand, z.B. eine Objektbewegung, die über Bewegungssensoren beobachtet werden kann. Die Kontextserver, in denen die Modellzustände festgehalten werden, dienen deshalb dem Ereignisdienst als Ereignisquelle (*Event Source*). Der Ereignisdienst selbst enthält zwei Komponenten (siehe Abb. 2.6) :

Observation Service ist für eine verteilte Beobachtung der Ereignisse auf den Ereignisquellen zuständig. Er besteht aus mehreren Beobachtungsknoten (*Observation Nodes*), die geschickt platziert werden, um ein verteiltes Ereignis zu beobachten.

Notification Service ist für die Benachrichtigung über eingetretene Ereignisse zuständig. Eine Anwendung muss sich beim Notification Service anmelden, wenn sie über das Eintreten eines Ereignisses benachrichtigt werden will.

Ist eine verteilte Ereignisbeobachtung durch den Observation Service nicht nötig, so kann ein Ereignis durch eine Anwendung auch direkt auf den Ereignisquellen registriert werden. Da die Beobachtung von einer Vielzahl von Ereignissen sehr rechenintensiv ist, wird die Beobachtung eines Ereignisses nicht automatisch mit der Registrierung gestartet. Es gibt deshalb folgende

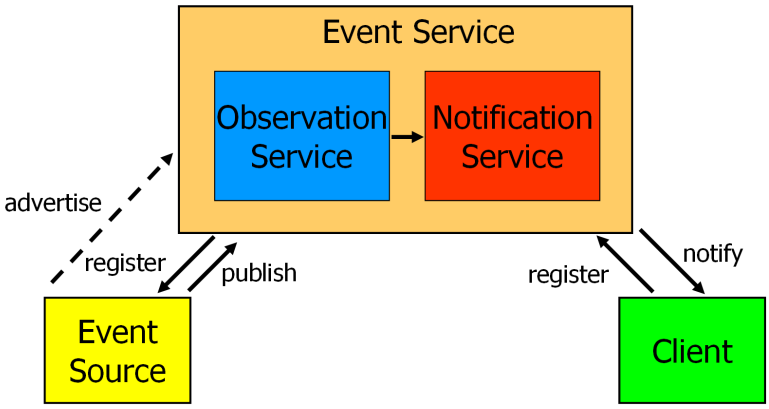

Abbildung 2.6: Event Service Architektur [Bau03]

zwei Phasen:

1. In der **Registrierungsphase** registriert die Anwendung beim Ereignisdienst das zu beobachtende Ereignis. Dabei wird das Ereignis durch Prädikate beschrieben, die in der Registrierungsnachricht mitgesendet werden. Prädikate sind u.a. der Ereignistyp und die dafür erforderlichen Parameter. *OnEnterArea (Gebiet, Person)* steht beispielsweise für das Ereignis, dass eine bestimmte Person ein angegebenes Gebiet betritt. Die erfolgreiche Registrierung wird in einer Bestätigungsnachricht mitgeteilt. Nach der Registrierung ist eine explizite Anmeldung der Anwendung beim Notification Service für dieses Ereignis nötig. Sobald das Ereignis eintritt, werden vom Notification Service alle Anwendungen benachrichtigt, die ihr Interesse für das Ereignis mitgeteilt haben.
2. Die Beobachtung eines Ereignisses muss aber in der **Observierungsphase** zuerst explizit durch die Anwendung eingeleitet werden. Während der Observierungsphase wird vom Ereignisdienst laufend berechnet, ob das Ereignis eingetreten ist oder nicht. Jedes Mal, wenn das Ereignis eingetreten ist, wird vom Notification Service eine Benachrichtigung an die Anwendung gesendet.

Bislang dienen dem Event Service als Ereignisquellen nur Location Server, die die Beobachtung von Ereignissen lokal durchführen. Eine Erweiterung auf Spatial Model Server ist vorgesehen (siehe auch [BDG+04]).
Soll in einer Workflowanwendung die Funktionalität des Ereignisdiensts bereitgestellt werden, so muss das asynchrone Verhalten des Ereignisdiensts und zusätzlich die Verteilung der Funktionalität in den verschiedenen Komponenten berücksichtigt werden. Benachrichtigungen werden beispielsweise nicht direkt von der Ereignisquelle, sondern vom Benachrichtigungsdienst versendet und der Versand erfolgt erst, sobald ein Ereignis eintritt.

2.1.6 Value Added Services

Als Mehrwertdienste (*Value Added Services*) werden sämtliche Dienste auf der Nexus Plattform bezeichnet, die die Funktionalität der Nexus Plattform erweitern, für die Funktionalität der zuvor vorgestellten Komponenten aber nicht erforderlich sind. Diese Dienste werden sozusagen auf die Plattform aufgesetzt, haben meist eine proprietäre Schnittstelle oder sind sogar unabhängig von den übrigen Diensten auf den Nexus Nodes. Darunter sind Dienste wie z.B. ein Navigationsdienst (*Navigation Service*), der Fahrtrouten berechnet, ein Kartendienst (*Map Service*), mit dem Karten

von bestimmten Gebieten erzeugt werden können, oder der *GeoCast Service*, mit dem man Nachrichten in bestimmte Gebiete versenden kann.
Bislang greift nur der Kartendienst auf den Datenbestand eines Kontextservers zu und die übrigen zwei Dienste arbeiten unabhängig von den Diensten der Nexus Plattform. Ein Konzept für eine erweiterte Version des Navigationsdiensts basierend auf dem Datenbestand eines Kontextservers ist bereits vorhanden (siehe [FHV06]). Im Hauptteil der Arbeit wird der Kartendienst als Value Added Service für eine Integration in Workflowumgebungen betrachtet (siehe Kapitel 4).

2.2 XML und XSL

Die *Extensible Markup Language* [XML] hat sich zu einem bedeutenden Standard in der Informationstechnik entwickelt. Das Ziel bei der Entwicklung von XML war es, ein einheitliches Format für den Austausch von Daten in webbasierten Anwendungen zu schaffen, weshalb XML auch als Basis für die Web Service Technologie genutzt wird (siehe Abschnitt 2.3). Mittlerweile wird XML jedoch auch von einer Vielzahl konventioneller Anwendungen verwendet. Die Eigenschaften die dazu führten und die Bestandteile von XML werden im Abschnitt 2.2.1 erklärt. XML ist, wie der Name bereits sagt, erweiterbar. Diese Eigenschaft wird beispielsweise von der *Extensible Stylesheet Language* (XSL) aufgegriffen, um eine Sprache für die Transformation und Formatierung von XML Dokumenten zu schaffen. Die Gründe, die für eine Transformation von Dokumenten sprechen, und die Komponente in XSL, die für die Transformation verwendet wird, wird im Abschnitt 2.2.2 erläutert. Für den Zugriff auf XML Dokumente, z.B. für die Selektion bestimmter Dokumentteile, wird schließlich im letzten Abschnitt die XPath Spezifikation vorgestellt.

2.2.1 XML

XML ist eine Spezifikation des *World Wide Web Consortium* (W3C) [XML], die ursprünglich für das Gebiet des Elektronischen Publizierens entworfen wurde, d.h. für eine groß angelegte Aufbereitung und Bereitstellung von Informationen im Internet. Mittlerweile wird XML auch in zahlreichen weiteren Anwendungsbereichen eingesetzt. Gründe dafür sind lt. [McL01]:

- **Erweiterbarkeit**: Ein Kernkonzept von XML ist deren Erweiterbarkeit. Aufbauend auf XML können eigene Sprachen und Datenformate definiert werden. XML beschränkt somit den Benutzer nicht auf festgelegte Strukturen bei der Formatierung eines Dokuments, wie dies z.B. bei HTML der Fall ist. Beispiele für die Erweiterung von XML sind die Spezifikationen XSLT (siehe Abschnitt 2.2.2), SOAP (siehe Abschnitt 2.3.1), WSDL (siehe Abschnitt 2.3.2) und BPEL (siehe Abschnitt 2.4.2).
- **Portabilität**: XML Daten können auf verschiedene Plattformen übertragen werden und somit ist eine Nutzung von XML Daten plattformunabhängig. Dies liegt daran, dass XML als einfacher Text repräsentiert werden kann. Zusätzlich kann in einem XML Dokument die Zeichencodierung angegeben werden. Durch die Spezifikation des W3C wird lediglich ein grober Rahmen für die Syntax und Semantik einzelner Bestandteile von XML vorgegeben (s.u.). Dies führt zu einer hohen Unterstützung durch zahlreiche Entwicklungswerkzeuge und Plattformen.
- **Interoperabilität**: Auf Basis von XML können, aufgrund der Portabilität und Erweiterbarkeit, unterschiedliche Systeme miteinander verbunden werden.

Listing 2.1: XML Beispieldokument

```
<mitarbeiter>
    <!-- Jeder Mitarbeiter hat
         eine eindeutige Personalnummer -->
    <personalNummer>12345</personalNummer>
    <name>Max Muster</name>
    <gewicht einheit="kg">75</gewicht>
</mitarbeiter>
```

Die fundamentalen Elemente eines XML Dokuments werden durch die *XML Infoset* Spezifikation festgelegt [InfoSet]. XML Infoset soll einen konsistenten Zugriff auf Informationseinheiten eines XML Dokuments in anderen Spezifikationen und Systemen fördern, indem es die Elemente der Informationsmenge (*information set*) festlegt, die weitergereicht werden können. Systeme können auf dieser Informationsmenge arbeiten. Wesentliche Informationseinheiten in XML Infoset sind:

- *Document*: Enthält die Information zum gesamten Dokument. Über diese Informationseinheit kann auf sämtliche weiteren Informationseinheiten zugegriffen werden. Diese Informationseinheit erscheint im Gegensatz zu den übrigen Informationseinheiten nur einmal in der Informationsmenge eines XML Dokuments.
- *Element*: Enthält die Information zu jedem Element eines XML Dokuments.
- *Attribute*: Enthält die Information zu jedem Attribut, das in einem Element vorkommen kann.
- *Character*: Enthält Zeichen eines Datensatzes, der in XML Dokumenten vorkommenden Daten.
- *Comment*: Enthält Informationen eines Kommentars im XML Dokument.

Für den Austausch von Informationseinheiten erfolgt eine Serialisierung der Informationsmenge, bekannt u.a. durch die Syntax der spitzen Klammern „< >". In Listing 2.1 ist ein Beispiel für eine Serialisierung einer Informationsmenge aufgeführt. Dabei ist „*mitarbeiter*" ein Element, welches die weiteren Elemente „*personalNummer*", „*name*" und „*gewicht*" enthält. Ein Element wird dabei durch einen beginnenden und einen schließenden Bezeichner definiert, wie z.B. beginnend mit <*mitarbeiter*> und schließend mit <*/mitarbeiter*>. Ein Element kann u.a. weitere Elemente, Zeichen eines Datensatzes, Kommentare oder Attribute enthalten. Ein Attribut wird innerhalb eines Elements in Form eines Paares angegeben, das durch die Angabe eines Bezeichners und eines Wertes definiert ist, wie z.B. beim Element <*gewicht einheit="kg"*> die Einheit des Gewichts in Kilogramm spezifiziert wird. Ein XML Dokument enthält genau ein Element, welches sämtliche Elemente umschließt (s.o. Informationseinheit Document). Im angegebenen Beispiel ist es das Element <*mitarbeiter*>.
Um die Struktur von XML Dokumenten festlegen zu können, z.B. dass ein Mitarbeiter aus der Personalnummer, dem Namen und dem Gewicht zusammengesetzt ist, wurde vom W3C die *XML Schema* Spezifikation [XMLSchema] entwickelt. XML Schema basiert ebenfalls auf der XML Syntax und ermöglicht neben der Strukturierung von Dokumenten auch die Definition von Datentypen der ausgetauschten Daten. Listing 2.2 zeigt das entsprechende XML Schema, das zur Definition der Struktur und der Datentypen für das zuvor genannte XML Dokument verwendet wird. Das *document* Element *mitarbeiter*, also das Element welches alle weiteren Elemente umschließen soll, setzt sich aus den drei Elementen *personalNummer*, *name* und

Listing 2.2: XML Schema zu Beispiel 2.1

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:MitarbeiterSchema">
  <xs:element name="mitarbeiter">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="personalNummer" type="xs:positiveInteger"/>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="gewicht">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:positiveInteger">
                <xs:attribute name="einheit" type="xs:string"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

gewicht zusammen. Für Letztere wurden auch einfache Datentypen angegeben, die ebenfalls von XML Schema zur Beschreibung von Daten zur Verfügung gestellt werden. Dabei werden für die Personalnummer und das Gewicht die positiven Ganzzahlen (*positiveInteger*) und für den Namen eine Zeichenkette (*string*) verwendet. Weitere Datentypen die von XML Schema bereitgestellt werden sind in Abbildung 2.7 dargestellt. Diese Datentypen können zur Definition neuer komplexer Datentypen verwendet werden. Ein Beispiel für die Definition komplexer Datentypen ist der Typ für das *mitarbeiter* Element, eingeschlossen durch den Bezeichner *complexType*. Der komplexe Typ für einen Mitarbeiter besteht folglich aus der Abfolge (*sequence*) von den o.g. drei Elementen, die wiederum auf den einfachen Datentypen von XML Schema basieren. Im Schema des Mitarbeiters aus Listing 2.2 wird ein weiteres wesentliches Element von XML verwendet, der Namensraum (*namespace*). Namensräume werden definiert, um eine eindeutige Zuordnung von Elementen in XML Dokumenten zu erreichen. Folgendes Beispiel: In einem Unternehmen gibt es die Personalabteilung und die Fertigung. Beide Bereiche legen für einen Mitarbeiter ein Schema an, in dem sie jeweils abteilungsspezifische Daten modellieren. Dabei wird zusätzlich in beiden Schemata für den Mitarbeiter der gleiche Bezeichner gewählt, z.B. *„mitarbeiter"*. Innerhalb der Schemata werden anschließend eindeutige Namen für das Schema selbst vergeben. Eine dritte Abteilung kann jetzt die bereits definierten Mitarbeitermodelle in einem neuen Schema verwenden, ohne sie dabei neu modellieren zu müssen. Sie referenziert die Elemente einfach über den Namen des Schemas, auch als *Namensraum* des Schemas bekannt. Werden beide Mitarbeitermodelle gleichzeitig im neuen Schema verwendet, so können sie zusätzlich über sog. Präfixe unterschieden werden, die dem Namensraum vorangestellt werden. Namensräume werden durch das Attribut *xmlns* referenziert bzw. mit dem Attribut *targetNamespace* neu vergeben. In Listing 2.2 sind zwei Namensräume angegeben. Zum einen wird mit *xmlns* der Namensraum von XML Schema eingebunden und zum anderen wird mit targetNamespace der Namensraum für das aktuelle Schema vergeben. Für den Namensraum von XML Schema wird im Beispiel zusätzlich das Präfix *xs* verwendet, um die Elemente von XML Schema eindeutig von den Elementen des

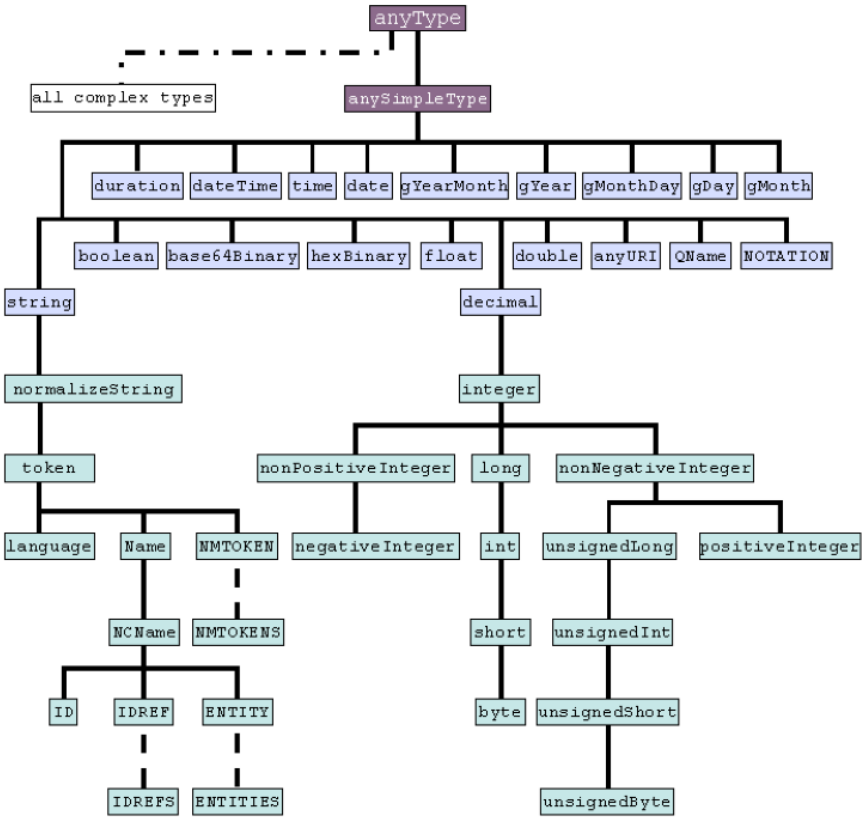

Abbildung 2.7: XML Schema Datentypen [XMLSchema]

neu erstellten Schemas zu unterscheiden. Wird ein Namensraum ohne Präfix eingebunden, also nur mit *xmlns*, so ist dies der Standardnamensraum eines Dokuments und alle Elemente ohne Präfix werden diesem Namensraum zugeordnet.
Aus dem Schema in Listing 2.2 lassen sich folglich Dokumentinstanzen erzeugen, die der definierten Struktur eines Mitarbeiters entsprechen. Die Dokumentinstanzen beinhalten dabei Daten, die den im Schema definierten Datentypen entsprechen. Ein Beispiel für eine Dokumentinstanz ist in Listing 2.1 angegeben.
Vor allem Workflowanwendungen gehören zu den zahlreichen Anwendungsbereichen, in denen für die Definition und den Austausch von Datentypen die XML Spezifikation genutzt werden kann. Workflowanwendungen schätzen die Vorteile der Erweiterbarkeit und Interoperabilität dieses Austauschformats. Außerdem sind auf der Nexus Plattform aus Gründen der Interoperabilität und der Integration neuer Anwendungen und Systeme u.a. das Augmented World Schema und die Schnittstellen zu den Nexusdiensten mit XML Schema beschrieben.

2.2.2 XSLT

Im vorherigen Abschnitt wurde gezeigt, dass sich XML besonders gut für das Repräsentieren von Daten eignet. In vielen Bereichen, wie z.B. dem Elektronischen Publizieren, ist zudem eine nachträgliche Aufbereitung und Formatierung der XML Daten für unterschiedliche Zwecke sinnvoll. Wird beispielsweise aus den Mitarbeiterdaten aus Listing 2.1 lediglich der Name benötigt und soll dieser Datensatz zugleich als HTML Seite in einem Browser aufgerufen werden

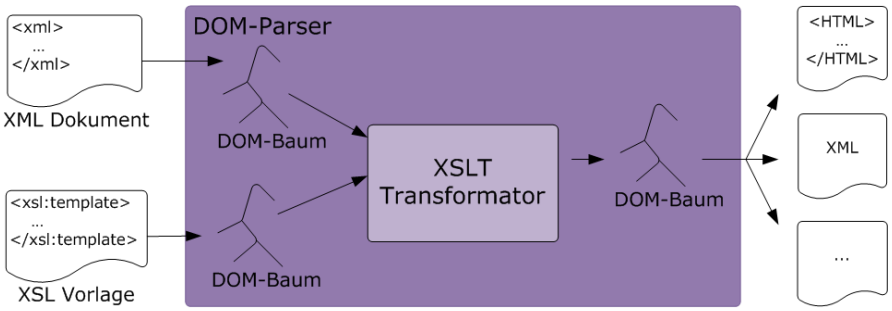

Abbildung 2.8: Funktionsweise eines XSLT Prozessors [HZ03]

können, so ist die zusätzliche Transformation und Formatierung der Daten erforderlich. Für diesen Zweck wurde *XSL Transformations* [XSLT] entwickelt.
Der Vorteil einer separaten Spezifikation für die Transformation von XML Dokumenten ist nach [CCD+01] eine erhöhte Flexibilität in der Aufbereitung der Daten für unterschiedliche Zwecke. Daten können zum einen in verschiedene Formate umgewandelt werden, wie z.B. HTML, Wireless Markup Language oder VoiceXML. Dies erlaubt eine Anpassung des Datensatzes an unterschiedliche Systeme und Anwendungen. Zum anderen können neue Sichten auf Daten erstellt werden, in denen durch Weglassen oder Einfügen bestimmte Daten ausgeblendet oder zusätzlich angezeigt werden. In Abbildung 2.8 ist die Funktionsweise eines XSLT Prozessors dargestellt. Ein XML Dokument und eine XSL Vorlage (*stylesheet*) dienen dem XSLT Prozessor als Eingabedaten. Dabei sind die Anweisungen zur Transformation des XML Dokuments in der XSL Vorlage spezifiziert. Der XSLT Prozessor nimmt diese Anweisungen und wendet sie entsprechend auf das XML Dokument an. Dazu erstellt er sich intern eine Baumstruktur vom XML Dokument und von den Anweisungen in der XSL Vorlage. Das *document* Element des XML Dokuments steht dabei in der Wurzel des Baumes und die untergeordneten Elemente entsprechen den Blättern entlang der Kanten. In der Baumstruktur der Vorlage existieren für jeden Blattknoten des XML Dokuments entsprechende Transformationsanweisungen für die entsprechenden Elemente des XML Baumes. In Listing 2.3 ist ein *XSL Stylesheet* aufgeführt, mit dem ein XSLT Prozessor aus dem Beispieldokument aus Listing 2.1 eine HTML Seite erstellt, die eine Liste mit den Mitarbeiterdaten enthält. Die wesentlichen Elemente aus dem XSLT Namensraum sollen kurz erläutert werden:

- Mit dem *stylesheet* Element wird erklärt, dass es sich bei diesem Dokument um ein Stylesheet für einen XSLT Prozessor handelt. Es wird die XSLT Version und der Namensraum der XSLT Spezifikation angegeben.
- Das *output* Element gibt an, von welchem Format die Ausgabe ist. Neben *html* gibt es z.B. auch *text* oder *xml*.
- Mit *template* Elementen können die Transformationsanweisungen für die Elemente des Quelldokuments definiert werden. Durch das Attribut *match* in template wird dabei das Element des Quelldokuments ausgewählt, für das diese Anweisung gelten soll. Für die Auswahl des Elements wird dabei XPath verwendet (siehe Abschnitt 2.2.3). In Listing 2.3 wird im ersten template Element mit dem XPath Ausdruck „/" die Wurzel des Quelldokuments ausgewählt. Weitere templates sind für die Elemente *personalNummer*, *name* und *gewicht* definiert worden.
- Mit *apply-templates* können an beliebigen Stellen eines Stylesheets weitere Transforma-

Listing 2.3: Stylesheet zur Erzeugung einer HTML Seite aus XML Dokument 2.1

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>
   <xsl:template match="/">
     <html><head><title>Mitarbeiterdaten HTML Seite</title></head>
       <body><h1>Mitarbeiterdaten</h1>
         <ul><xsl:apply-templates/></ul>
       </body>
     </html>
   </xsl:template>
   <xsl:template match="personalNummer">
     <li>Personalnummer: <xsl:value-of select="."/></li>
   </xsl:template>
   <xsl:template match="name">
     <li>Name: <xsl:value-of select="."/></li>
   </xsl:template>
   <xsl:template match="gewicht">
     <li>Gewicht: <xsl:value-of select="."/><xsl:value-of select="@einheit"/></li>
   </xsl:template>
</xsl:stylesheet>
```

tionsanweisungen aufgerufen werden. Dabei kann im Attribut *select* ein spezielles template gewählt werden. Im Beispielstylesheet 2.3 fehlt das select Attribut, weshalb an dieser Stelle alle templates angewendet werden, die im Stylesheet vorhanden sind.

- Das Element *value-of* erlaubt die Ausgabe der Zeichen (Text), die durch ein Element eingeschlossen sind. Die Auswahl des Elements, für das die Ausgabe erfolgen soll, geschieht ebenfalls durch einen XPath Ausdruck in einem select Attribut.

Es gibt weitere Elemente in XSLT, die bei der Transformation von Dokumenten hilfreich sind, wie z.B. *for-each* für die iterative Bearbeitung von Elementen oder *copy-of* für das Kopieren ganzer Teile einer Baumstruktur. Für eine detaillierte Beschreibung siehe [XSLT].
Für die Integration der Nexusdienste in Workflowumgebungen können Transformationen u.a. für eine Abbildung eines einfachen Workflow-Nachrichtenformats auf ein komplexeres Nexus-Nachrichtenformat verwendet werden (siehe Kapitel 4).

2.2.3 XPath

Für die Adressierung von Informationseinheiten in XML Dokumenten wurde die *XML Path Language* [XPath] entwickelt. Für XSLT ist XPath deshalb ein wesentlicher Bestandteil zur Anwendung von Transformationsanweisungen auf Elemente, Attribute, Kommentare oder Text. Grundsätzlich erfüllen XPath Ausdrücke folgende Zwecke in XSLT [CCD+01]:

- Die Auswahl von Knoten für die Verarbeitung.
- Die Aufstellung von Bedingungen für eine differenzierte Bearbeitung von Quelldokumenten.
- Erstellung von Text, der im Zieldokument eingefügt werden soll.

Für die Lokalisierung von Dokumentbestandteilen wird in XPath das Konzept des *location path* verwendet. Der location path gibt, ähnlich einer Anfahrtsbeschreibung zu einem Gebäude, den Weg zu einem Element im XML Dokument an. Folgendes Beispiel: Eine Person wird auf der Straße nach dem Weg zu einem markanten Punkt in einer Stadt gefragt. Daraufhin wird die Person anfangen, den Weg zu beschreiben. Hierfür kann sie z.B. die Wegweiser *rechts*,

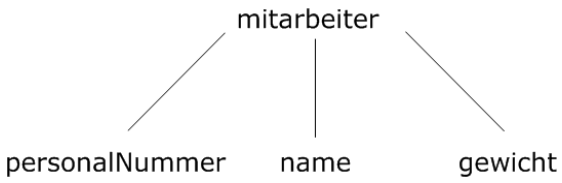

Abbildung 2.9: Baumstruktur für Element *mitarbeiter*

links oder *geradeaus* verwenden oder sie verwendet Straßennamen und markante Plätze, die auf dem Weg liegen, wie z.B. ein Kino oder eine breite Allee. Dies wird in XPath ebenfalls aufgegriffen durch die Konzepte des *absolute* und *relative location path*. Die Pfadangabe basiert dabei auf der Baumstruktur des Quelldokuments (siehe Abb. 2.9). Der absolute Pfad auf die Personalnummer wird durch *mitarbeiter/personalNummer* angegeben. Für den relativen Pfad bedient sich XPath seines Kontexts. Dabei ist XPath die aktuelle Position in der Baumstruktur eines Dokuments bekannt, z.B. während der Abarbeitung eines Dokumentbaumes durch einen XSLT Prozessor. Dann ist es wie in Listing 2.3 möglich, nur *personalNummer* als Pfad für das gewünschte Element anzugeben. Für die Bewegung durch die Dokumentstruktur gibt es in XPath, ähnlich der Wegbeschreibung durch *rechts* und *links*, 13 verschiedene Achsen, darunter *self* für die aktuelle Position im Baum, *child* für die Knoten unterhalb der aktuellen Position oder *parent* für den Knoten, der über der aktuellen Position steht. Somit ermöglicht XPath die Navigation durch die komplette Dokumentstruktur und die Auswahl von Dokumentteilen. Zusätzlich bietet XPath zahlreiche Funktionen, welche die Arbeit erleichtern, wie z.B. die Funktion *count()* zum Abzählen von Elementen. Weitere Informationen finden sich bei [XPath].

2.3 Serviceorientierte Architektur mit Web Services

Die Bereitstellung von Funktionalität in einer IT-Landschaft in Form von Dienstleistungen (*services*), ermöglicht im Vergleich zu der bisherigen Form der Module, Funktionen oder Pakete eine neue Ebene der Abstraktion. Services werden abstrakt beschrieben und zu einem vorher vereinbarten Preis und einer bestimmten Qualität angeboten. Ähnlich einem Energieversorger, der seine Dienstleistung „Strom" beschreibt und Kunden zu einem vorher festgelegten Preis anbietet, wird beispielsweise von einem Internetanbieter ein Dienst zur Sortierung von Daten zu einem vereinbarten Preis angeboten.
Diese neue Form der Dienstleistung benötigt eine neue Architektur, eine serviceorientierte Architektur, die das Finden und Anbieten von Diensten ermöglicht und eine einfache Einbindung der Dienste unterstützt. Neben den Rollen eines Dienstanbieters (*service provider*) und eines Dienstnutzers (*service requestor*) ist zum Auffinden von Diensten ein Dienstverzeichnis (*discovery facility*) hilfreich. Diese Konstellation ist in Abbildung 2.10 dargestellt. Die Realisierung einer serviceorientierten Architektur wurde erfolgreich durch die Web Service Technologie vorangetrieben. Die Architektur ist aus mehreren Schichten aufgebaut (siehe Abb. 2.11). Die Transportschicht (*Transport*) regelt den Versand von Daten über Kommunikationsleitungen, z.B. über das HTTP Protokoll. Auf der Nachrichtenschicht (*Messaging*) werden Daten als Nachrichten abstrahiert. Spezifikationen wie XML, SOAP und WS-Addressing bilden hierbei die Grundlage für den Nachrichtenaustausch zwischen den sog. *Web Services* (siehe Abschnitt 2.3.1). Auf der Beschreibungsschicht (*Description*) finden sich Spezifikationen für die abstrakte Beschreibung von Diensten. Dabei ist WSDL (siehe Abschnitt 2.3.2) für die funktionale

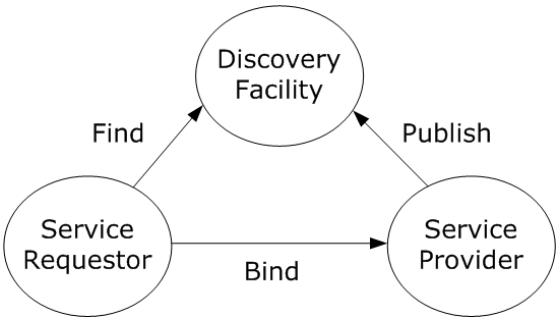

Abbildung 2.10: SOA Dreieck [WCL+05]

Beschreibung der Dienste zuständig. WS-Policy wird für zusätzliche Forderungen zu einem Dienst verwendet, wie z.B. Sicherheitsrichtlinien zur Verschlüsselung der Daten beim Versand. Die Dienstgüteschicht (*Quality of Service*) hält für die Beschreibung zusätzlicher Qualitätseigenschaften von Diensten weitere Spezifikationen bereit, wie z.B. den zuverlässigen Versand von Nachrichten (WS-ReliableMessaging), Sicherheit (WS-Security) oder Transaktionen (WS-Atomic Transaction, WS-Business Activity). Auf der obersten Ebene (*Components*) finden sich dann als Komponenten entweder eigenständige Dienste (Atomic) oder zusammengesetzte Dienste (Composite) wieder. Für zusammengesetzte Dienste, d.h. Dienste die beispielsweise aus mehreren atomaren Web Services zusammengestellt sind, wird die Prozessbeschreibungssprache BPEL definiert (siehe Abschnitt 2.4.2). Neben diesen Schichten ist die Verwaltung der Dienste durch ein Dienstverzeichnis sinnvoll. Dafür wird z.B. die Spezifikation *Universal Description and Discovery Interface* (UDDI) zur Verfügung gestellt, die das Anbieten und Finden von Diensten ermöglichen soll.
Serviceorientierte Architekturen bieten neben der losen Kopplung und dem Auffinden von Diensten zusätzlich die Möglichkeit eines modularen Systemaufbaus. Ein System, das auf der Web Service Architektur basiert, kann z.B. für den Transport von Daten HTTP verwenden, für den Versand der Nachrichten JMS nutzen, die Beschreibung des Diensts mit WSDL durchführen, als Dienstgüte die sichere Übermittlung von Daten mit WS-Security unterstützen und seine Dienste als atomare Einheiten zur Verfügung stellen. Spätere Erweiterungen des Systems durch zusätzliche Funktionen, wie z.B. durch Transaktionalität, können auf dem vorhandenen System aufsetzen. Die Integration der Nexusdienste in Workflowumgebungen soll im Hauptteil dieser Arbeit auf Basis der Bestandteile einer Web Service Architektur erfolgen. Hierzu werden die Nexusdienste u.a. für den Versand über SOAP vorbereitet und durch abstrakte Beschreibungen ergänzt. Die WS-Addressing Spezifikation soll beispielsweise zur Verarbeitung von Ereignissen der Nexus Plattform verwendet werden. In den folgenden Abschnitten werden deshalb die Spezifikationen WSDL und SOAP näher betrachtet. Es wird eine Implementierung der SOAP Spezifikation vorgestellt, die eine Vielzahl der Komponenten der Web Service Architektur unterstützt. Schließlich wird im Abschnitt 2.4.2 die BPEL Spezifikation erläutert, mit der die Möglichkeit gegeben wird, Workflows zu beschreiben.

2.3.1 SOAP

Die Kommunikation zwischen Web Services durch SOAP-Nachrichten [SOAP] bietet wesentliche Vorteile:

- Austausch einer standardisierten Nachrichtenstruktur auf Basis von XML.

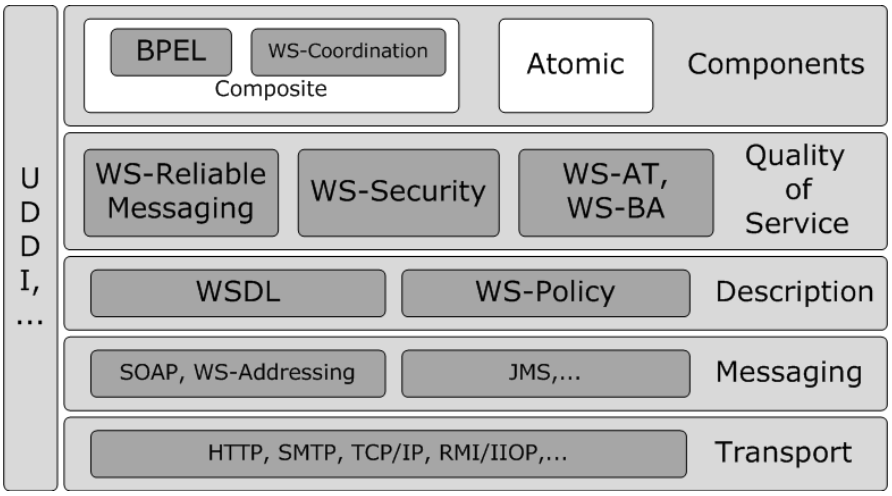

Abbildung 2.11: Web Service Architektur [WCL+05]

- Die Nutzung von SOAP bietet Interoperabilität zwischen verschiedenen Plattformen.
- SOAP Nachrichten können über beliebige Transportprotokolle versendet werden, wie z.B. TCP/IP oder HTTP.
- Durch SOAP Nachrichten werden die Funktionalitäten der Web Service Architektur unterstützt, wie z.B. Sicherheit durch WS-Security (s.u.).

Eine SOAP Nachricht beinhaltet sämtliche Informationen, die für eine Interaktion mit einem Web Service benötigt werden. Dabei können beim Versand einer SOAP Nachricht zwischen dem Sender und dem endgültigen Empfänger beliebig viele Nachrichtenempfänger liegen. Die letztere Eigenschaft ist allein aufgrund der Strukturen von Netzwerken sinnvoll. Abbildung 2.12 zeigt die wesentliche Einteilung einer SOAP Nachricht.

Envelope Dieses Element definiert den Beginn einer SOAP Nachricht und umschließt somit sämtliche Bestandteile der Nachricht selbst.

Header Die Definition von Header Elementen hat eine wesentliche Bedeutung. Hier werden Verarbeitungsanweisungen für die Empfänger einer SOAP Nachricht definiert. Eine Verarbeitungsanweisung kann dabei Elemente einer Spezifikation der Web Service Architektur repräsentieren. In einem WS-Security Header werden beispielsweise Parameter übertragen, die von der WS-Security Spezifikation für die Erfüllung sicherheitsrelevanter Aktionen vorgegeben sind, wie z.B. in einfachster Weise eine Benutzerkennung und ein Passwort. SOAP Header unterstützen auch die Erweiterung der Web Service Architektur. Werden neue Spezifikationen eingeführt, so kann deren Funktionalität durch die Definition der entsprechenden SOAP Header ermöglicht werden. Die Bearbeitung der Header durch die verschiedenen Empfänger einer SOAP Nachricht wird dabei durch die Zuweisung von Rollen gesteuert.

Body In diesem Teil der Nachricht befinden sich die eigentlichen Daten, die zwischen zwei Anwendungen ausgetauscht werden sollen. Diese werden vom endgültigen Empfänger einer SOAP Nachricht erwartet und dienen der Verarbeitung auf Anwendungsebene. Der Body kann beispielsweise die Eingabedaten für einen Funktionsaufruf enthalten.

Fault Fehler in der Verarbeitung werden aufgrund ihrer Wichtigkeit bei der Kommunikation

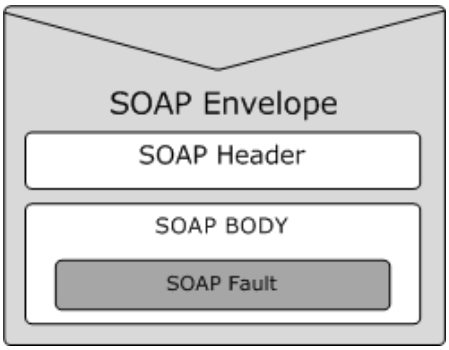

Abbildung 2.12: Aufbau einer SOAP Nachricht [Mel07]

zwischen lose gekoppelten Diensten als eigenständige Elemente behandelt. In einem SOAP Fault können Meldungen über aufgetretene Fehler übertragen werden. Gründe für Fehlermeldungen wären z.B. ein falsches Datenformat, fehlende Parameter oder Netzwerkprobleme. Ein SOAP Fault wird innerhalb eines Body Elements verschickt.

Weiterführende Informationen zur SOAP Spezifikation finden sich bei [SOAP].

2.3.2 WSDL

Der Nachrichtenaustausch zwischen Web Services wird durch SOAP ermöglicht. Für die Übermittlung einer SOAP Nachricht an einen Web Service fehlt jetzt noch die wesentliche Definition dessen, was überhaupt übermittelt werden soll bzw. kann. Diese Funktion übernimmt die *Web Service Definition Language* [WSDL]. WSDL Beschreibungen teilen sich in einen abstrakten Teil auf, in dem definiert wird, was eine Web Service Schnittstelle kann, und in einen konkreten Teil, in dem festgelegt wird, wie und wo diese Funktionalität angeboten wird. In WSDL Version 1.1 wird der abstrakte Teil durch die Elemente *types*, *message*, *operation* und *portType* beschrieben, das *wie* des konkreten Teils durch das *binding* Element und das *wo* durch die Elemente *port* und *service*. In Abbildung 2.13 ist der Aufbau einer WSDL Beschreibung dargestellt. Das Element operation befindet sich innerhalb der portType Deklaration und das Element port innerhalb der service Deklaration.

types In diesem Element ist die Definition von Datentypen möglich, die in den Nachrichten ausgetauscht werden sollen. Für die Definition einfacher und komplexer Datentypen kann XML Schema verwendet werden (siehe Abschnitt 2.2.1).

message Hier ist eine abstrakte Definition der Nachrichten möglich, die zwischen Web Service Schnittstellen ausgetauscht werden sollen. Die Nachrichten basieren auf den Datentypen die im types Bereich deklariert werden.

operation Hier erfolgt eine abstrakte Beschreibung der Funktionalität, die ein Web Service anbietet. Dafür werden für ein *operation* Element die Ein- und Ausgabenachrichten (*input* und *output*) angegeben, die im *message* Bereich definiert wurden. Da die Nachrichten auf einem bestimmten Datentyp basieren, ist die Definition vergleichbar mit der Deklaration einer Funktionssignatur bei klassischen Programmiersprachen. Das wesentliche im operation Element ist die Anordnung der Nachrichten. Wird nur die input Nachricht angegeben, so handelt es sich um eine Operation, bei der keine Rückgabe erwartet wird, auch *one-way* Operation genannt. Wird nur ein output Element angegeben, so sendet der Service

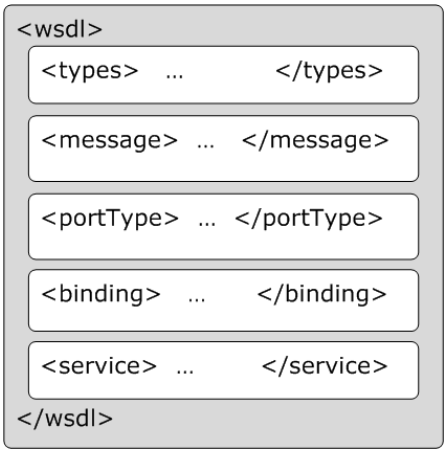

Abbildung 2.13: Aufbau einer WSDL Beschreibung [WCL+05]

aktiv eine Nachricht ohne eine Eingabe dafür zu erwarten, auch *notification* Operation genannt. Wird ein input Element vor einem output Element deklariert, so handelt es sich um einen Aufruf des Services, bei dem zusätzlich eine Rückgabe erwartet wird, auch als *request-response* Operation bezeichnet. Schließlich wird im umgekehrten Fall der Service eine Nachricht an die Anwendung versenden und eine Antwort erwarten, was als *solicit-response* Operation bezeichnet wird [WCL+05].

portType Dieses Element fasst eine Menge von Operationen zusammen, die von einem Web Service Endpunkt angeboten werden. Es ist ähnlich der Schnittstellenbeschreibung bei klassischen Programmen.

binding Mit dem binding Element wird festgelegt, wie die zuvor abstrakten Definition konkret umgesetzt werden sollen. Hier wird das Transportprotokoll, z.B. HTTP, und das Nachrichtenformat, z.B. SOAP, festgelegt. Wird SOAP als Nachrichtenformat gewählt, so werden hier die SOAP-Header angegeben, die für den Nachrichtenaustausch benötigt werden und der Nachrichtenstil, der beim Versand gewählt wird. Als Nachrichtenstile werden hauptsächlich *RPC/Encoded* und *Document/Literal* eingesetzt. RPC/Encoded steht für einen methodenhaften Aufruf, bei dem die Parameter und der Funktionsname angegeben werden, wie es bei klassischen Funktionen der Fall ist. Gleichzeitig wird zur Kodierung der Daten SOAP Encoding verwendet. Document/Literal hingegen steht für den Versand von kompletten Dokumenten, die durch XML Schema beschrieben sind. Da hier kein Methodenname übergeben wird, muss die gerufene Schnittstelle anhand des übergebenen Dokuments feststellen, was zu tun ist. Dafür erfolgt aber die Definition und Kodierung der Nachrichten einheitlich durch XML.

port Dieses Element gibt den Endpunkt eines Web Services an, der die im binding Element definierten Operationen implementiert. Dabei wird z.B. die URL Adresse eines Servlets angegeben, falls der Transport über HTTP erfolgt.

service Dieses Element fasst eine Menge von Endpunkten zusammen, die im port Bereich definiert wurden. Über das service Element kann folglich eine Web Service Schnittstelle an mehreren Endpunkten auf unterschiedliche Weise zugegriffen werden.

Zusätzliche Informationen zu den Beschreibungsmöglichkeiten einer Web Service Schnittstelle mit WSDL finden sich bei [WSDL].

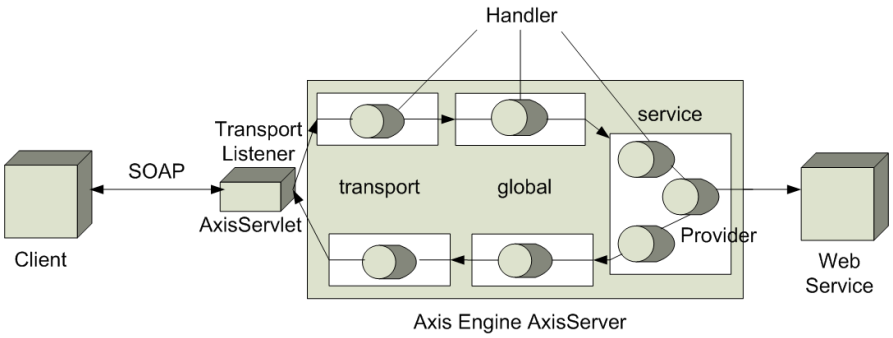

Abbildung 2.14: Apache Axis Architektur [WBFT04]

Apache Axis Für viele Spezifikationen aus der Web Service Architektur gibt es bereits zahlreiche Implementierungen. Apache Axis [Axis] ist eine Implementierung der SOAP Spezifikation (siehe Abschnitt 2.3.1) und bietet darüber hinaus weitere Funktionen. In Abbildung 2.14 ist die Architektur von Apache Axis Version 1.4 abgebildet. Axis unterstützt dabei den Anwender u.a. bei der Erstellung von SOAP Aufrufen durch entsprechende Hilfsklassen, bei der Codegenerierung von Server- und Klientenklassen für die Bereitstellung eigener Web Service Endpunkte, bei der Serialisierung und Deserialisierung von Datentypen, bei der Analyse von Nachrichten durch Monitoring-Werkzeuge oder beim Versand der Nachrichten über verschiedene Transportprotokolle, wie z.B. HTTP. Die Generierung von Programmcodegerüsten kann anhand von WSDL Beschreibungen erfolgen. Ebenso kann eine WSDL Beschreibung anhand bereits existenter Programmschnittstellen erstellt werden.

Für den Empfang von Nachrichten über verschiedene Transportprotokolle werden unterschiedliche Endpunkte bereitgestellt. Für den Empfang von SOAP Nachrichten über HTTP wird ein Servlet bereitgestellt, welches Aufrufe an den Web Service entgegennimmt und an die entsprechende Implementierung des Web Services weiterleitet (siehe Bild). Vor dem Aufruf der Web Service Implementierung wird eine Kette von Verarbeitungspunkten durchlaufen, sog. *Handler Chains*. Ein Handler realisiert dabei das Konzept der SOAP Header. Mit SOAP Headern können verschiedene Spezifikationen der Web Service Architektur realisiert werden (siehe Abschnitt 2.3.1). Für die Unterstützung der Spezifikationen können in Axis also bestimmte Handler eingesetzt werden, z.B. ein Handler, der die WS-Addressing Spezifikation implementiert. Handler können dabei verschiedenen Bereichen zugeteilt werden, dem Transportbereich, dem globalen Bereich oder dem dienstspezifischen Bereich. Folglich können Handler beispielsweise im Transportbereich nur dem HTTP Protokoll zugeordnet werden und werden dann auch nur verwendet, wenn Nachrichten über dieses Transportprotokoll versendet werden. Der letzte Handler, der für den Aufruf der Web Service Implementierung zuständig ist, wird als Provider bezeichnet. Für verschiedene Web Service Implementierungen, z.B. für einfache Java Klassen, für Enterprise Java Beans oder für Skripte, existieren unterschiedliche Provider, die den Aufruf durchführen.

2.4 Workflows

Die Betrachtung und Analyse von Prozessabläufen in einem Unternehmen gewinnt immer mehr an Bedeutung. Viele Unternehmen schätzen den Wert dieser Geschäftsprozesse ähnlich hoch ein wie den Wert der traditionellen Geschäftsgüter Maschinen, Kapital und Arbeitskraft [LR00]. Die Abbildung der Geschäftsprozesse in ein Prozessmodell ermöglicht durch dessen Abstraktion

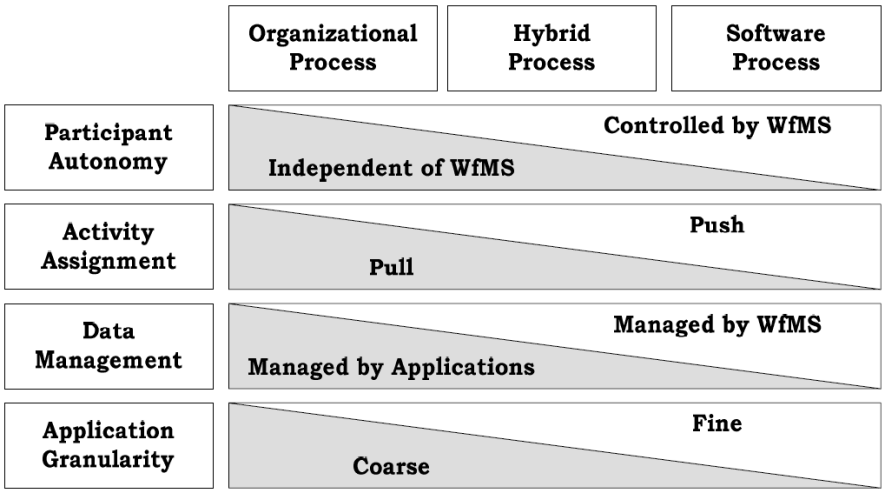

Abbildung 2.15: Klassifikation von Workflowsystemen [JMG02]

ein besseres Verständnis für die Abläufe. Für die Teile eines Prozessmodells, die sich durch informationstechnische Mittel unterstützen lassen, können separate *Workflowmodelle* erstellt werden. Workflowmodelle konzentrieren sich also lediglich auf die Bereiche eines Gesamtprozesses, in denen Software eingesetzt wird [JBS97]. Auf Basis dieser Modelle lassen sich informationstechnische Systeme erstellen ([JBS97],[LR00]) und Optimierungen von Prozessabläufen durchführen [Dav93]. Für Workflowmodelle werden Metamodelle definiert, durch die eine syntaktische und semantische Beschreibung der Workflowmodelle erfolgen kann. Verschiedene Metamodelle finden sich bei [LR00] oder bei [WfMC]. Einzelne Arbeitsabläufe, sog. Workflowinstanzen, folgen dann schließlich dem vorgeschriebenen Ablauf eines Workflowmodells. Die Workflowinstanzen werden dabei von einem Workflowmanagementsystem verwaltet, das die korrekte Einhaltung des Ablaufes anhand des Modells und des Metamodells gewährleisten soll. Im weiteren Verlauf dieser Arbeit wird für die Begriffe Workflowmodell und Workflowinstanz der einheitliche Begriff *Workflow* verwendet.
Für die Beschreibung und Ausführung von Workflows, die sich hauptsächlich durch eine Komposition von Web Services Aufrufen auszeichnen, wurde die *Business Process Execution Language* Spezifikation [BPEL] entworfen. Im Hauptteil der Arbeit wird BPEL für die Beschreibung der Workflows verwendet, in denen eine Integration von Kontextinformationen erfolgen soll. Deshalb werden die Elemente von BPEL in einem Unterabschnitt kurz erläutert. Zuvor wird eine Klassifikation von Workflows und Workflowsystemen gegeben.

2.4.1 Klassifikation von Workflows

Eine bekannte Klassifikation von Workflows wurde von der Giga Information Group verwendet, einem Marktforschungsunternehmen in der Informationstechnik, das mittlerweile in das Forrester Unternehmen integriert wurde. Diese Klassifikation wird in der Literatur u.a. von [LR00] oder [All01] aufgegriffen und definiert die folgenden Arten von Workflows:

- *Production*: Bei diesem Workflow steht die Automatisierung von einer großen Anzahl ähnlicher Tätigkeiten im Vordergrund. Workflows dieser Art repräsentieren zusätzlich die wesentlichen Produktionsprozesse eines Unternehmens. Ein manueller Eingriff wird nur in Ausnahmefällen erwartet.

- *Administrative*: Dieser Workflow greift Tätigkeiten aus der Verwaltung auf. Hier steht die einfache Erstellung von neuen Prozessdefinitionen im Vordergrund. Dabei werden neue Prozesse meist über Formulare bereitgestellt, in die ein Mitarbeiter Daten eingeben kann, wie z.B. die Daten für die weitere Bearbeitung eines Arbeitsvertrags.
- *Collaborative*: Bei diesem Workflow liegt das Hauptaugenmerk auf der Unterstützung von Gruppen bei der Ausführung von gemeinsamen Tätigkeiten. Systeme, die durch diese Workflowart gekennzeichnet sind, gehören meist auch zur Kategorie „*groupware*“.
- *Adhoc*: Adhoc-Workflows zeichnen sich dadurch aus, dass sie dem Benutzer ermöglichen, auf einfache und schnelle Weise neue Prozessdefinitionen zu erstellen. Bei diesen Workflows kann es durchaus vorkommen, dass für einen Großteil der Prozessdefinitionen nur jeweils eine Prozessinstanz vorkommt, d.h. deren Ausführungshäufigkeit ziemlich gering ist.

Zusätzlich werden bei dieser Klassifikation die Workflowtypen entsprechend dem Wert, den sie für ein Unternehmen haben und der Anzahl der Wiederholungen, mit denen sie ausgeführt werden, eingeordnet. Dabei werden *Production Workflows* sehr oft in gleicher Form ausgeführt und haben zusätzlich einen sehr hohen Stellenwert für ein Unternehmen.
Eine Klassifikation von workflowbasierten Systemen wird von [JMG02] gegeben und ist in Abbildung 2.15 zu sehen. Beim organisatorischen Typ werden bestimmte Personen oder Komponenten benachrichtigt, dass Arbeit zur Erledigung bereitsteht (siehe *workitems* [LR00]). Außerdem werden eventuell die nötigen Anwendungen aufgerufen, mit denen der Benutzer die Aufgabe erledigen kann. Der Workflow ist sich aber nicht über den Inhalt der Anwendungen bewusst, d.h. der Anwender muss sich um die korrekte Ausführung der Aufgabe kümmern. Als Beispiel für den organisatorischen Typ könnte man die Benachrichtigung des Systems über die Konstruktion eines bestimmten Fahrzeugteiles mit einem gleichzeitig gestarteten CAD-Programm nehmen. Beim Software-Typ verknüpft der Workflow die Komponenten verschiedener Anwendungen auf Mikroebene, d.h. es werden Funktionen von Anwendungen aufgerufen, denen die Parameter zur Ausführung übergeben werden. Als Beispiel könnte man ein nächtliches Update bestimmter Datenquellen nehmen. Der hybride Typ versucht Legacy-Systeme in den Softwareprozess mit einzubinden. Als Beispiel könnte man eine Weboberfläche nehmen, welche die Eingaben von Benutzern über Formulardaten entgegennimmt und eine vorhandene Anwendung zur Berechnung der Versicherungssumme einer Lebensversicherung aufruft.

2.4.2 BPEL

In der Web Service Architektur (siehe Abb. 2.11) kann zur Komposition von Web Services die *Business Process Execution Language* [BPEL] verwendet werden. Die BPEL Spezifikation entstand ursprünglich aus der Verbindung der existierenden Prozessbeschreibungssprachen *WSFL* [WSFL] von IBM und *XLANG* [XLANG] von Microsoft. Im Jahr 2003 wurde dem OASIS Standardisierungskomitee [OASIS] *BPEL for Web Services* Version 1.1 [BPEL] zur Standardisierung vorgeschlagen. BPEL4WS basiert dabei auf den Standards XML, WSDL und XPath (s.o.). Bei OASIS wurde die BPEL Spezifikation schließlich Anfang dieses Jahres in einer erweiterten Version unter dem Namen WS-BPEL 2.0 standardisiert. Diese Arbeit basiert, aufgrund der bisherigen Etablierung in zahlreichen Systemen und Anwendungen, auf der BPEL4WS 1.1 Spezifikation und im weiteren Verlauf wird dafür der Name BPEL verwendet.
In Listing 2.4 ist die Struktur einer BPEL Beschreibung auf oberster Ebene aufgeführt. Ein Workflow, der mit BPEL beschrieben wird, wird vom *process* Element eingeschlossen. Weitere

wesentliche Elemente aus der BPEL Spezifikation können in die folgenden Bereiche eingeteilt werden [Mel07]:

Datenverarbeitung Die Trennung der Daten vom Kontrollfluss wird durch die Definition eines *variable* Elements im *variables* Bereich ermöglicht. Hier werden die Daten spezifiziert, die im Prozess verarbeitet werden sollen. Diesen Variablen kann innerhalb eines Prozessablaufs mit dem *assign* Element ein Wert zugewiesen werden.

Aktivitäten Aktivitäten repräsentieren den Kontrollfluss in einem Prozess. In Aktivitäten werden die Daten aus den Variablen verwendet. Mit einem *invoke* Element wird beispielsweise ein Web Service aufgerufen. Mit *receive* und *reply* Elementen stellt der Prozess seine Funktionalität einem Aufrufer nach außen hin zur Verfügung, indem er auf eine Anfrage wartet oder ein Ergebnis zurück liefert. Aktivitäten lassen sich auch zusammensetzen. Mit dem *sequence* Element wird beispielsweise eine Folge von Aktivitäten eingeschlossen. Mit den Elementen *switch* und *while* lassen sich Aktivitäten unter bestimmten Bedingungen gruppieren und mit der Aktivität *flow* können parallele Aktivitäten beschrieben werden. Wenn Aktivitäten innerhalb der Bereiche der Ereignisbehandlung, Ausnahmebehandlung bzw. Kompensation auftreten, dann haben sie für den Prozess eine spezielle Bedeutung (s.u.).

Kommunikation Zur Kommunikation mit anderen Web Services wird das *partnerLink* Element verwendet. Das partnerLink Element wird im Bereich *partnerLinks* definiert und von den Aktivitäten verwendet, die weitere Web Services aufrufen, z.B. invoke. Das partnerLink Element ist dabei verbunden mit der WSDL Beschreibung eines Web Services (siehe Abschnitt 2.3.2). Außerdem werden sowohl dem Aufrufer als auch dem aufgerufenen Web Service Rollen vergeben, die von beiden während eines Aufrufs eingenommen werden. In einem invoke Element wird dann beispielsweise dem Prozess die Rolle des Aufrufers zugewiesen und dabei die Schnittstelle und die Operation des aufgerufenen Web Services angeben. Der Web Service selbst erhält die Rolle des Dienstanbieters.

Ereignisbehandlung Für die Behandlung von Ereignissen, wie z.B. dem Eintreffen einer Benachrichtigung, ist im Bereich *eventHandlers* die Deklaration der Elemente *onMessage*, *onAlarm* und *pick* möglich. Im Element onMessage wird auf die Nachricht gewartet, im Element onAlarm wird nach Ablauf einer bestimmten Zeit eine Aktivität ausgeführt und im pick Element lassen sich mehrere Ereignisse zusammenfassen, wobei auf genau eines der Ereignisse gewartet wird. Ereignisse werden vor allem bei Abläufen unbestimmter Dauer eingesetzt.

Korrelationsmengen Für einen Workflow können mehrere Instanzen existieren. Die Zuordnung eines Ergebnisses aus einem Web Service Aufruf, z.B. einer Bestellung, muss zu der entsprechenden Workflowinstanz erfolgen, z.B. zu der Instanz, die von einem bestimmten Benutzer aufgerufen wurde. Zu diesem Zweck können im Bereich *correlationSets* mit dem Element *correlationSet* solche Zuordnungen definiert werden. Die Zuordnungen werden dabei auf den Daten definiert, die dem Prozess übergeben werden oder zur Verfügung stehen, z.B. einer Benutzerkennung. In den Aktivitäten zur Kommunikation mit den Web Services (invoke, receive, reply, onMessage) werden diese Zuordnungen dann verwendet.

Ausnahmebehandlung Zur Behandlung von Fehlern und zur Wiederherstellung von vorherigen Zuständen stehen die Bereiche *faultHandlers* und *compensationHandlers* zur Verfügung. Hier können Aktivitäten definiert werden, die im Falle eines bestimmten Fehlers durchgeführt werden sollen. Die Kompensation von Aktivitäten erfolgt zumeist im Kontext von Transaktionen. Hier sollen bereits erfolgreich durchgeführte Aktivitäten

Listing 2.4: Struktur einer BPEL Beschreibung [Mel07]

```
<process name="ProzessName">
    <partnerLinks>          ... <partnerLinks>
    <variables>             ... <variables>
    <correlationSets>       ... <correlationSets>
    <faultHandlers>         ... </faultHandlers>
    <compensationHandlers>  ... </compensationHanlders>
    <eventHandlers>         ... </eventHandlers>
    <!-- mindestens eine Aktivität -->
</process>
```

wieder rückgängig gemacht werden, d.h. z.B. eine erfolgreich durchgeführte Buchung soll durch den Aufruf einer Stornierung kompensiert werden. Fehlerhaft ausgeführte Aktivitäten werden nicht kompensiert, sondern durch eine Fehlerbehandlung abgefangen, z.B. durch den Abbruch des Prozesses bei der Meldung eines Fehlers in der Kommunikation zum Web Service.

Weiterführende Informationen zur BPEL Spezifikation finden sich bei [BPEL].

3 Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten im Bereich kontextsensitiver Umgebungen präsentiert. Es werden deshalb neben der Nexus Plattform (siehe Abschnitt 2.1.1) zuerst weitere Implementierungen traditioneller kontextsensitiver Umgebungen vorgestellt. Da das Interesse in Bezug auf *Service- und Prozessorientierung* bei der Bereitstellung kontextsensitiver Daten und der Erstellung kontextbezogener Anwendungen mittlerweile gestiegen ist (siehe Einleitung), werden zusätzlich Systeme vorgestellt, die gezielt auf die Realisierung dieser Forderungen eingehen. Zu jedem System werden dabei die Gründe aufgeführt, die schließlich zur Wahl der Nexus Plattform für die Bereitstellung von Kontextinformationen geführt haben.

3.1 Ausgewählte Beispiele traditioneller kontextsensitiver Systeme

Im Bereich der kontextsensitiven Umgebungen gibt es zahlreiche traditionelle Architekturen, in denen nicht oder nur in geringem Umfang auf die vorhandenen Standards serviceorientierter Architekturen, wie z.B. der Web Service Architektur, zur Bereitstellung von Kontextinformationen eingegangen wird. Drei traditionelle Kontextsysteme sollen hier kurz vorgestellt werden und es sollen die Vorteile der Nexus Plattform gegenüber diesen Systemen aufgeführt werden.

3.1.1 Context Toolkit

Context Toolkit [SDA99] ist, wie der Name schon sagt, eine Sammlung von Schnittstellen und Funktionen, die dem Programmierer zur Erstellung von kontextsensitiven Anwendungen zur Verfügung gestellt werden. Die Context Toolkit Architektur besteht aus drei wesentlichen Komponenten (siehe Abb. 3.1):

- **Widgets** werden als vermittelnde Einheiten zwischen der Umgebung und der Anwendung verwendet. Dabei werden Daten der Umgebung über Sensorik aufgenommen und über eine einheitliche Schnittstelle zur Verfügung gestellt.
- **Aggregators** sind eine Erweiterung der Widgets. Sie haben zusätzlich die Fähigkeit, einfachere Daten, die von den Widgets bereitgestellt werden, zu komplexeren Daten zu aggregieren. Somit können sie für die Repräsentierung von komplexeren Objekten wie z.B. Personen oder Gebäuden genutzt werden.
- **Interpreters** sollen als zusätzliche Komponenten einfache Daten, die von Widgets bereitgestellt werden, auswerten und entsprechend interpretieren, z.B. die Interpretation eines Meetingbeginns nach Abdunklung eines Raums und gleichzeitigem Start eines Projektors.

Für Anwendungen werden Aggregator und Interpreter Komponenten erstellt, die dann für den Zugriff auf Kontextinformationen verwendet werden. Als Kontextmodell dienen Attribut-Werte-Paare, wie z.B. die Temperatur entspricht 24 Grad Celsius. Context Toolkit eignet sich für die

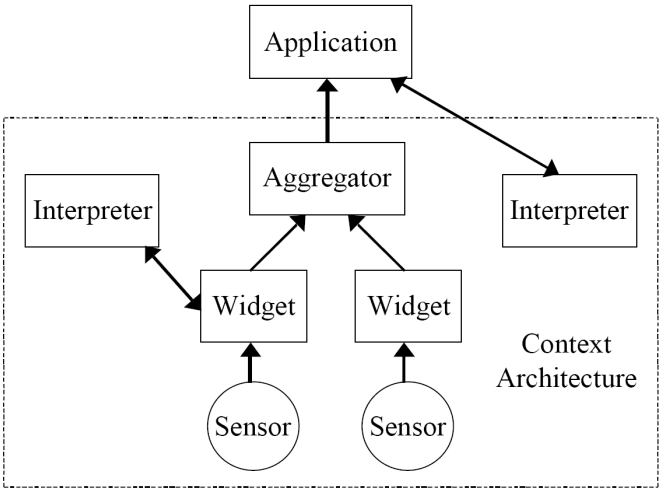

Abbildung 3.1: Beispielarchitektur einer Anwendung in Context Toolkit [SDA99]

Erstellung einfacher Anwendungen, die einen speziellen Zweck erfüllen, wie z.B. eine Tafel, die anzeigt, welche Mitarbeiter sich zur Zeit im Gebäude befinden.
Bei [SC06] wird festgestellt, dass die Intelligenz bei Context Toolkit in der Aggregation von Kontextdaten und dem Absenden geeigneter Anfragen an die Kontextdatenbank steckt. Dies ist auch bei der Nexus Plattform der Fall. Im Vergleich zu Context Toolkit bietet die Nexus Plattform jedoch den Vorteil, dass auch komplexe Anwendungen erstellt werden können. Dafür stellt die Nexus Plattform ein Umgebungsmodell bereit, das Augmented World Model (siehe Abschnitt 2.1.2), das sich durch seine Erweiterbarkeit und Skalierbarkeit auszeichnet. Gleichzeitig bietet die Föderation die Möglichkeit, hochwertige Aufbereitungsfunktionen für die Daten der Augmented World zu definieren.

3.1.2 EasyMeeting

Für intelligente Konferenzräume (sog. smart meeting rooms) wird in [CFJ+04b] die EasyMeeting Umgebung vorgestellt, mit der die Arbeit von mehreren Personen an gemeinsamen Zielen unterstützt werden soll, wie z.B. bei der Abhaltung eines Meetings oder bei der Vorführung einer Präsentation. Es handelt sich um ein kontextbasiertes System, welches auf der *Context Broker Architecture* [CFJ04a] (CoBrA) aufbaut. CoBrA bietet ein zentralisiertes Kontextmodell, welches von einem zentralen Agenten, dem ContextBroker (Broker-Agent) über weitere verteilte Agenten des Frameworks (z.B. Sensor-Agenten) zusammengesucht und letztlich durch Schlussfolgerungen über die gesammelten Daten aufbereitet wird. Sensor-Agenten erfassen dabei Informationen über Sensoren, wie z.B. die Helligkeit im Zimmer. Folgendes Szenario könnte man sich für ein Meeting vorstellen: Die Teilnehmer einer Präsentation betreten den Präsentationsraum. Dies wird dem Broker-Agent von einem Sensor-Agent mitgeteilt. Der Broker-Agent führt anschließend entsprechende Aktionen aus, wie z.B. die Abdunklung des Raumes oder den Start der Präsentation.
Der ContextBroker ist, wie bereits erwähnt, die zentrale Komponente in CoBrA und besteht aus einer Wissensdatenbank, einer „Reasoning“-Engine, einem Modul zur Beschaffung von Kontextinformationen und einem Modul für die Rechteverwaltung der Agenten.
Die Nexus Plattform basiert auf einem skalierbaren Umgebungsmodell, das auf Servern abgelegt

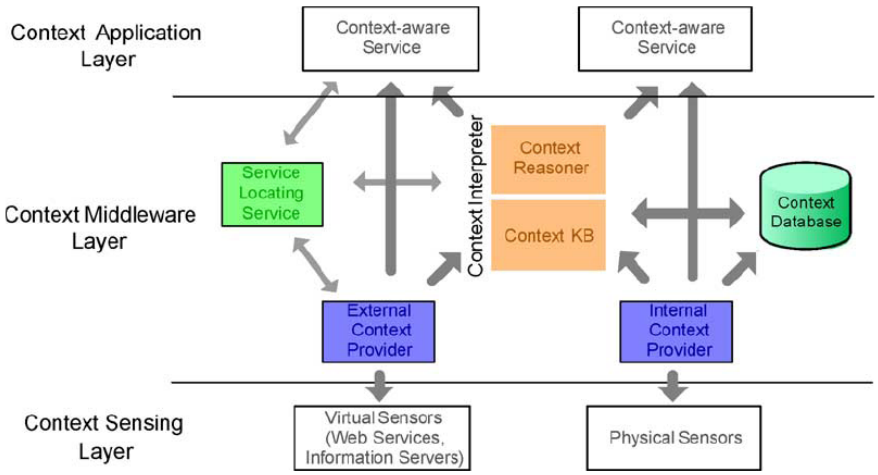

Abbildung 3.2: SOCAM Architektur [GPZ05]

ist, die für eine hohe Anzahl von Anfragen ausgelegt sind. Dies ist sinnvoll für eine Produktionsumgebung, in der eine Vielzahl von Prozessen auf einen gemeinsamen Datenbestand zugreift. Dabei sollen die Prozesse, anstatt von Agenten, von einem stabilen Workflowsystem verwaltet werden. Der zusätzliche Management- und Kommunikationsaufwand zwischen den Agenten, der die eigentliche Bearbeitungszeit von Anfragen erhöht, fällt dadurch weg.

3.1.3 SOCAM

In [GPZ05] wird als eine weitere Architektur zur Bereitstellung von Kontextinformationen die *Service Oriented Context Aware Middleware* vorgestellt. Die Architektur hat drei verschiedene Ebenen (siehe Abb. 3.2). Auf der Anwendungsebene befinden sich die Anwendungen, die auf die Kontextdaten der darunter liegenden Ebene zugreifen. Auf der mittleren Ebene, der Kontext-Middleware-Ebene, liegen die wesentlichen Dienste, die sich um die Bereitstellung und Aufbereitung der Daten kümmern. Dazu zählen der Context Provider, der nützliche Informationen aus heterogenen Informationsquellen extrahiert, der Context Interpreter, der eine logische Auswertung dieser Daten vornimmt, und der Service Locating Service, mit dem die kontextsensitiven Anwendungen der obersten Ebene die vorhandenen Context Provider und Context Interpreter finden können, um über deren Schnittstellen auf Kontextinformationen zuzugreifen. Auf der untersten Ebene, werden die Kontextdaten über Sensoren erfasst. Dabei werden virtuelle Sensoren, wie z.B. ein Wetterdienst im Internet, von physischen Sensoren, wie z.B. Sensoren zur Temperaturmessung, unterschieden. SOCAM basiert auf einem Ontologie-Modell, d.h. Daten müssen zuerst anhand von Ontologien beschrieben werden, damit sie vom Context Interpreter und Context Provider erkannt und ausgewertet werden können. Die Anwendungen sprechen die Komponenten des Context Middleware Layer über proprietäre Schnittstellen an. Das Hauptaugenmerk liegt auf dem Schlussfolgern über die gefundenen Daten und der Aufbereitung der Daten. SOCAM ist ein zentralisierter Ansatz, bei dem der Context Interpreter die zentrale Komponente ist. Die Kommunikation mit den Komponenten erfolgt über Java-RMI.
Die logische Verarbeitung von Daten erfolgt bei ontologiebasierten Systemen durch die Definition von Regeln auf den Datensätzen und mittels Schlussfolgerungen. Regeln müssen somit mit Bedacht gewählt werden, da sie die erfasste Datenmenge reduzieren und sich so auf die Qualität

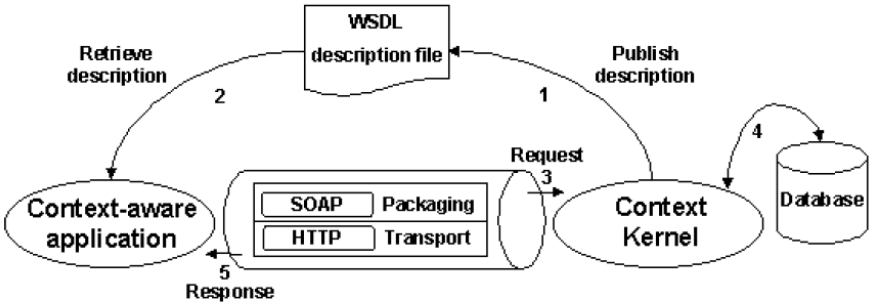

Abbildung 3.3: Context Kernel Architektur [ANGP03]

der Resultate auswirken. Auf der Nexus Plattform wird dies durch intelligente Anfragen erreicht. Die Qualität der Resultate kann somit auf der Nexus Plattform sowohl durch Aufbereitungsfunktionen auf der Föderationsschicht als auch durch eine individuelle Aufbereitung in der Anwendungsschicht erfolgen.

3.2 Ausgewählte Beispiele prozess- und serviceorientierter Kontextsysteme

In [BD04] wird auch darauf hingewiesen, dass traditionelle Frameworks bislang ihre eigenen Nachrichtenformate zur Kommunikation mit den Anwendungen verwenden und eine standardisierte Lösung wünschenswert wäre, z.B. auf Basis einer serviceorientierten Architektur mit Web Services (siehe Abschnitt 2.3). In diesem Abschnitt sollen Systeme zur Bereitstellung von Kontextinformation vorgestellt werden, die Bestandteile einer serviceorientierten Architektur oder eines Workflowsystems integrieren und somit die Forderung nach zusätzlicher *Service- und Prozessorientierung* beim Zugriff auf Kontextinformationen erfüllen. Dabei wird besonders auf die Verwendung existierender Standards, wie z.B. der Web Service Architektur oder der BPEL Technologie, geachtet.

3.2.1 ContextKernel

In [ANGP03] wird vorgeschlagen, für die Bereitstellung von Kontextinformationen die Web Service Technologie einzusetzen. Dabei wird die Architektur aus Abbildung 3.3 verwendet. Der *Context Kernel* wurde für den Einsatz in Capture&Access Anwendungen entwickelt. Capture&Access-Anwendungen sind dabei Office-Anwendungen, deren Hauptziele die Erfassung und der Zugriff von Daten sind, wie z.B. in einem elektronischen Notizbuch. Die Datenmodellierung erfolgt auf Basis der „*who, where, when, what, why and how*“ Dimensionen aus [TAB01]:

WHO: Wer darf Daten erfassen bzw. auf Daten zugreifen?

WHAT: Was wird erfasst bzw. auf was wird zugegriffen?

WHEN: Wann werden/wurden Daten erfasst bzw. wann werden/wurden die Daten zugegriffen?

WHERE: Wo geschah die Erfassung bzw. der Zugriff der Daten?

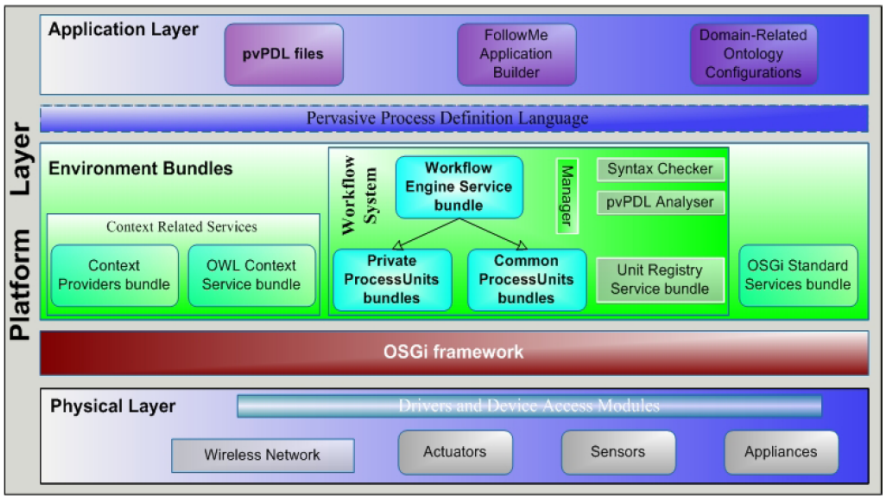

Abbildung 3.4: FollowMe Architektur [CBL+06]

HOW: Wie, d.h. mit welchen Mittel wurde die Erfassung bzw. der Zugriff durchgeführt?

Kontextdaten werden hier also bezüglich der Erfassung von Daten betrachtet, z.B. von *wem* wurde *wann* ein Text erstellt. Der Context Kernel ist als Web Service implementiert und bietet zum Speichern und Abrufen von Kontextdaten entsprechende Operationen an. Anwendungen nutzen die Schnittstelle des Context Kernel, um Daten, die in Form der Dimensionen beschrieben wurden, zu senden und um solche Daten auch abzurufen. Alle am ContextKernel angemeldeten Anwendungen können die Daten anderer Anwendungen abrufen.
Der ContextKernel wurde sowohl in *Computer Supported Collaborative Work Anwendungen* (CSCW) [NJCGGP04] als auch in Lernanwendungen [JNR05] eingesetzt und getestet. Folgende Vorteile wurden dabei u.a. genannt: asynchrone Verarbeitung, lose Kopplung und effizienter Datenaustausch:

> „*...web based infrastructures would not only store and retrieve, but also exchange information through the web.*“ [NJCGGP04]

Die Nexus Plattform bietet Anwendungen unterschiedlicher Bereiche die Möglichkeit, auf Kontextinformationen zuzugreifen und ist somit nicht auf Capture&Access Anwendungen beschränkt. Außerdem können komplexe Objekte modelliert werden, die sowohl Kontextinformationen darstellen (Objekte verhalten sich passiv, z.B. Gebäude) als auch neue Kontextinformationen erzeugen (Objekte verhalten sich aktiv, z.B. mobile Objekte die ihre Position ständig ändern).

3.2.2 FollowMe

FollowMe [CBL+06] ist eine *Workflow-Embedded Middleware*, die zur Unterstützung der Anwendungserstellung von kontextbasierten Anwendungen entwickelt wird. Das vorgeschlagene System beschreibt Daten durch Ontologien, enthält eine eigene *Workflowmaschine*, für die eine eigene Prozessbeschreibungssprache (pervasive Process Definition Language, pvPDL) definiert wurde und basiert auf dem OSGi Framework [OSGi]. Über das OSGi Framework werden die Komponenten zur Erfassung der Kontextdaten integriert, z.B. Sensormodule. Die Prozessbeschreibungssprache pvPDL basiert auf XML. Anfragen an FollowMe werden mittels RDQL

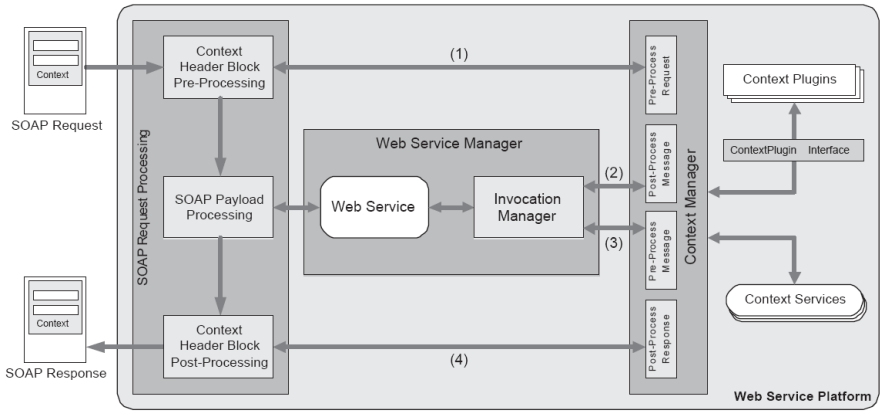

Abbildung 3.5: Komponenten bei der Kontextverarbeitung in Service Globe [KK04]

(RDF Data Query Language) formuliert. Mit RDF (Ressource Description Framework) lassen sich Daten um semantische Informationen in Form von Tripeln ergänzen. Die Workflowmaschine in FollowMe versteht RDQL und deshalb können Anfragen in RDQL direkt als Variablen in pvPDL Beschreibungen genutzt werden. Das Interessante an diesem System ist der Einsatz einer Workflowmaschine zur Erstellung von kontextsensitiven Anwendungen. In [CBL+06] wurde festgestellt, dass es effizienter ist, Anwendungen in Form von Workflows zu erstellen, indem einfache Services (die Kontextinformationen liefern), zu komplexeren Anwendungen zusammengesetzt werden. In Abbildung 3.4 ist die Architektur von FollowMe zu sehen, in die das Workflowsystem eingebettet ist.
Anstatt einer zusätzlichen Prozessbeschreibungssprache, wie sie bei FollowMe verwendet wird, sollen auf der Nexus Plattform etablierte Standards eingesetzt werden, wie z.B. BPEL. Dies erleichtert zum einen die Anwendungsentwicklung und zum anderen die Interoperabilität mit anderen Systeme. Die Dienste der Nexus Plattform werden dabei in die BPEL Prozesse für die Bereitstellung von Kontextinformationen integriert.

3.2.3 Service Globe

Mit *Service Globe* wird in [KK04] ein Framework vorgestellt, welches die Entwicklung und das Deployment von *kontextsensitiven Web Services* erleichtern soll. Dabei wird der Fokus darauf gelegt, dass in Zukunft Systeme unterschiedlicher Hardware auf die Dienste im Internet zugreifen werden. Der Kontext ist bei Service Globe also vor allem das Wissen über die Heterogenität der Systeme. Abbildung 3.5 zeigt die Kontextverarbeitung der Service Globe Plattform. Die Verarbeitung von Kontextinformationen geschieht in drei Komponenten, im aufgerufenen Web Service, durch zusätzliche Context Plugins (lokale Java Objekte) und durch zusätzliche Context Services (weitere Webdienste). Ein Kontexttyp wird im Header einer SOAP Nachricht versendet. Je nach Kontexttyp wird eine bestimmte Verarbeitung aufgrund der im Kontexttyp übergebenen Daten erfolgen. Folgendes Szenario: Eine Anwendung sendet eine Anfrage für ein bestimmtes Produkt, das erworben werden soll. Im Header der Nachricht wird, zusätzlich zu den Metadaten über das Anwendersystem, der Kontexttyp „Location“ mit den globalen Positionsdaten des Anfragenden mitgesendet. Der aufgerufene Web Service sucht nach Produkten, die der Anfrage entsprechen. Ein zusätzlicher Context Service „Routenplaner“ wird aufgrund der vorhandenen Kontextinformation „Location“ aufgerufen und ermittelt die Route zum Verkaufsstandort. Von

einem Context Plugin werden die Daten für die Rückgabe an das anfragende System (Handheld) in komprimierter Form aufbereitet. In [KK04] müssen für die richtige Ausführung der nötigen Context Plugins und Context Services Bearbeitungsanweisungen definiert werden (Context Processing Instruction). Diese können entweder im Header mitgesendet werden oder zusätzlich zum angebotenen Web Service in der Service-Registry definiert werden.
Die Nexus Plattform bietet im Vergleich zu Service Globe die Möglichkeit zur Modellierung und Bereitstellung von Kontextinformationen. Anwendungen der Nexus Plattform können sich somit nicht nur an den Kontext anpassen, sondern auch eigenen Kontext erstellen, z.B. durch die Erstellung von Objekten im Datenmodell. Die Auswahl von Kontextinformationen erfolgt auf diesem Datenmodell und geht somit weit über den einfachen Aufruf von Internetdiensten hinaus (wie z.B. den Aufruf einer Suchmaschine im WWW).

3.2.4 CAWE

Eine Framework für die kontextsensitive Verwaltung von Anwendungen basierend auf der Komposition von Web Services wird in [AFG+07] vorgestellt. Die Eigenschaften des *Context Aware Workflow Execution* Frameworks (CAWE) sind: die hierarchische Repräsentierung von Workflows mit *abstrakten Aktivitäten*, die Assoziierung von abstrakten Aktivitäten mit konkreten Implementierungen abhängig vom Kontext und die Spezifikation von Bedingungen zur Selektion der kontextabhängigen Teile eines Workflows. CAWE wurde auf einer Workflowmaschine aufgesetzt (jBPM), in der die Definition von abstrakten Aktivitäten möglich ist. Abstrakte Aktivitäten erlauben das Ausführen von beliebigem Programmcode bzw. die Definition weiterer Subprozesse. Die entsprechende Implementierung der abstrakten Aktivitäten wird anhand einer Datenbank ermittelt, in der die Beziehung der Kontextinformationen zu den jeweiligen Implementierungen festgelegt ist. Die Wahl für eine Implementierung könnte beispielsweise von der aufrufenden Person, dem aktuellen Standort oder der aktuellen Zeit abhängig gemacht werden. Folgende Vorteile werden aufgeführt:

- Die Unterstützung eines *top-down* Entwurfs von Anwendungen durch Abstraktion.
- Die Unterstützung der Vorgehensweise bei der Definition von Prozessen in Unternehmen durch eine hierarchische Repräsentierung eines Workflows, der auf einfacheren Workflows aufbaut.
- Eine verbesserte Lesbarkeit des kontextsensitiven Workflows.

Anhand der Kontextinformationen, die von der Nexus Plattform bereitgestellt werden, können in einer zukünftigen *BPEL Engine* ebenfalls Prozesse definiert werden, die anhand von Entscheidungskomponenten unterschiedliche Aktionen durchführen. Zur besseren Lesbarkeit werden dann, anstatt der abstrakten Aktivitäten, Subprozesse definiert. Die erwähnten Vorteile gelten im Allgemeinen für alle Workflowanwendungen, da Prozesse meist hierarchisch strukturiert werden.

4 Nexus in Workflows - Nutzung von Kontextdaten in BPEL Prozessen

In diesem Hauptteil der Arbeit wird die Integration von Kontextdaten in Workflowanwendungen anhand der Nexus Plattform realisiert. Der Hauptteil gliedert sich dabei in die klassischen Schritte, die beim Entwurf von Softwaresystemen durchgeführt werden: die Analyse, der Grob- und Feinentwurf, die Implementierung und die Verteilung der entworfenen Komponenten.
Für die Integration von Nexusdiensten in Workflows werden hier hierarchische *Integrationsprozesse* vorgestellt, die für den Zugriff auf die Nexusdienste auf einer serviceorientierten Architektur basieren und die Komplexität der Schnittstellen von Nexus durch Transformationen handhaben.
Die abstrakten Prozessmodelle werden mit entsprechenden BPEL Beschreibungen umgesetzt, die fehlenden Web Service Schnittstellen werden implementiert und es werden entsprechende XSLT Stylesheets für die Transformation von Nachrichten aufgeführt.
In einem abschließenden Beispiel soll anhand einer Workflowanwendung die Verwendung der im Hauptteil dieser Arbeit entworfenen *Integrationsprozesse* aufgezeigt werden.

4.1 Analyse und Anforderungen

In diesem Abschnitt wird die momentane Situation der Nexus Plattform analysiert. Dieser Schritt ist für die Identifizierung der Probleme wichtig, die später im Entwurf des Systems zur Integration von Nexusdiensten in Workflows beachtet werden sollen. Für den Entwurf sind dabei die unterschiedlichen Datentypen und Datenformate, die während der Kommunikation mit den Diensten ausgetauscht werden, und die dafür verwendeten Kommunikationsprotokolle von Interesse. Anschließend werden zusätzliche funktionale und nichtfunktionale Anforderungen für die Integration der Kontextdienste in eine Workflowumgebung erarbeitet.

4.1.1 Datentypen, Datenaustauschformate und Funktionalität in Nexus

Für den Zugriff auf die Funktionen der Nexus Plattform und die damit zusammenhängende Kommunikation mit den jeweiligen Diensten werden verschiedene Formate in Nexus spezifiziert. Die Formate beschreiben die erforderlichen Daten, die für einen ordentlichen Funktionsablauf dem Dienst übergeben werden müssen. Die Einhaltung und Kenntnis der Formate ist wesentlich für die Integration der Dienste in die Workflowumgebung. Deshalb werden die einzelnen Formate für die verschiedenen Dienste in den nachfolgenden Abschnitten kurz vorgestellt. Am Schluss wird auf die Probleme eingegangen, die sich durch die Formate ergeben. Alle Formate sind durch XML Schema beschrieben. Im Anhang A finden sich zusätzlich detaillierte Beschreibungen der Formate.
Die Kontextserver und die Föderation verwenden zum Austausch von Daten die gleichen Formate. Es werden dabei drei verschiedene Formate vorgegeben: die *Augmented World Query Language*

- *AWQL*, die *Augmented World Modeling Language - AWML* und die *Change Report Language - CRL* (siehe Anhang A.4). Das *AWQL* Format ist ein komplexeres Format, mit dem nicht nur die Abfrage von Daten möglich ist, sondern auch die Änderung. Das Kernelement in AWQL bildet eine Bedingung (*restriction*), mit der die Objekte der Augmented World ausgewählt werden können. Die Bedingung erlaubt eine verschachtelte, aussagenlogische Formulierung eines Wahrheitswerts, der auf der Nexus Plattform für die Auswahl eines Objekts entscheidend ist. Neben der Abfrage von Objekten definiert AWQL zusätzlich optionale *Filter* Elemente, mit denen die ausgewählten Objekte verändert werden können. Durch Filter können z.B. Objektattribute ausgeblendet, aktualisiert oder einem Objekt nachträglich hinzugefügt werden. Im AWQL Format werden zusätzlich Elemente für geografische Abfragen spezifiziert (siehe Anhang A.2), die dann z.B. zur Abfrage von Objekten in einem bestimmten Gebiet genutzt werden können. Zur Modellierung und zum Austausch von Objektdaten wird das *AWML* Format definiert. Das AWML Format ist ziemlich generisch gehalten, indem es lediglich den Austausch von abstrakten Objekten (*nexusobject*) und abstrakten Attributen (*nexusattribute*) vorschreibt. In einer AWML Beschreibung können somit sämtliche Objekte aus den Standardschemata der Augmented World auftauchen (siehe Anhang A.1). Die zusätzliche Zahl der möglichen Typen, die an Stelle der abstrakten Typen verwendet werden können, wird durch die Definition erweiterter Klassen und Attribute erhöht. Das einfache, generische Format wurde aus Gründen der Integration neuer Anwendungen in die Nexus Plattform gewählt. Mit der *Change Report Language* wird schließlich ein Format bereitgestellt, das Elemente zur Information über durchgeführte Änderungen bereitstellt. Die entsprechenden Kontextserver erstellen Rückgaben im CRL Format, in denen sie einer Anwendung den Erfolg (success) bzw. den Misserfolg (failure) einer Änderung mitteilen.
Der Ereignisdienst definiert für den Zugriff auf seine Funktionen und den Austausch von Informationen drei verschiedene Nachrichtenformate: die *Event Registration Language - ERL*, die *Event Notification Language - ENL* und *Event Information - EventInfo* (siehe Anhang A.6). Dabei sind im *ERL* Format die erforderlichen Daten für die Registrierung von Ereignissen festgelegt, die an den Ereignisdienst zur erfolgreichen Verarbeitung übergeben werden müssen. Erforderliche Daten sind z.B. die Angabe des Ereignistyps oder der Dauer für die Beobachtung des Ereignisses. Jeder Ereignistyp erwartet zusätzlich weitere spezifische Parameter. Eine Anwendung muss also für eine erfolgreiche Registrierung das ERL Format einhalten. Die Registrierung eines Ereignisses wird vom Ereignisdienst mit einem *EventInfo* Dokument bestätigt, das wesentliche Informationen enthält, wie z.B. einen eindeutigen Bezeichner oder die Zeit der Registrierung. Mit Dokumenten, die dem *ENL* Format entsprechen, erfolgen die Benachrichtigungen über eingetretene Ereignisse durch den Ereignisdienst. Das ENL Format definiert dazu verschiedene interessante Daten, wie z.B. die Beobachtungszeit oder den Beobachtungsort.
Für den Kartendienst existieren zwei Schnittstellenbeschreibungen: die *Map Predicate Language - MapPL* und die *Map Modeling Language - MapML* (siehe Anhang A.7). Eine Anwendung kann Aufrufe an den Kartendienst mit *MapPL* formulieren, um entsprechende Kartendaten zu erhalten. MapPL Anfragen bestehen dazu aus zwei wesentlichen Einheiten: einem Element zur Bestimmung von Objekten und weiterer Elemente zur Darstellung der ausgewählten Objekte. Für die Bestimmung der Objekte ist in MapPL Anfragen ein zusätzliches Dokument im AWQL Format enthalten (s.o.). Für die Darstellung ist in MapPL Anfragen z.B. die Angabe von Zeichenregeln, CSS-Formatierungen oder der Kartenauflösung möglich. Der Map Service liefert das Ergebnis im MapML Format, in dem die Bilder entweder als Vektor- oder Rastergrafiken beschrieben sind.
Tabelle 4.1 enthält eine Auflistung der vorhandenen Schemata, bestehend aus dem XML Namespace und dem Prefix, der üblicherweise verwendet wird. Zusätzlich ist das Element angegeben, das später zur Markierung eines Dokumenttyps als Wurzelelement verwendet wird.

XML Prefix	XML Namespace	Document Tag
NSAT	http://www.nexus.uni-stuttgart.de/1.0/NSAT	N/A
NSAS	http://www.nexus.uni-stuttgart.de/1.0/NSAS	N/A
NSCS	http://www.nexus.uni-stuttgart.de/1.0/NSCS	N/A
GML	http://www.opengis.net/gml	N/A
AWQL	http://www.nexus.uni-stuttgart.de/2.0/AWQL	<awql>
AWML	http://www.nexus.uni-stuttgart.de/2.0/AWML	<awml>
CRL	http://www.nexus.uni-stuttgart.de/2.0/CRL	<crl>
ERL	http://www.nexus.uni-stuttgart.de/1.0/ERL	<predicate>
EventInfo	http://www.nexus.uni-stuttgart.de/1.0/EventInfo	<EventInfo>
ENL	http://www.nexus.uni-stuttgart.de/1.0/ENL	<notification>
MAPPL	http://www.nexus.uni-stuttgart.de/2.0/MapPL	<mappl>
MAPML	http://www.nexus.uni-stuttgart.de/2.0/MapML	<mapml>

Tabelle 4.1: Übersicht Nexus Schemata

Probleme

Zusammenfassend ergeben sich folgende Schwierigkeiten durch die zuvor vorgestellten Schnittstellenbeschreibungen:

- **Komplexe Schnittstellen**: Die Nachrichtenformate in Nexus sind zum Teil sehr komplex. Ein Programmierer muss bislang viele Daten bereitstellen und sich mit den verschiedenen Strukturen der Nachrichtenformate auseinander setzen. Im AWQL Format wird beispielsweise durch das *choice* Element eine Auswahlmöglichkeit von mehreren Filtern geschaffen (siehe Anhang A.4), die eine Determinierung des Rückgabeformats im Workflow erschweren. Außerdem werden unterschiedliche Funktionalitäten in einem Nachrichtenformat gebündelt, z.B. die Abfrage und Änderung von Daten im AWQL Format oder die Registrierung unterschiedlicher Ereignistypen im ERL Format. Bestimmte Elemente haben einen rekursiven Aufbau, z.B. das *restriction* Element aus dem AWQL Format. Am *restriction* Element wird auch deutlich, dass in den Nachrichtenformaten nicht nur Datentypen beschrieben werden, sondern auch logische Verarbeitungsanweisungen. Im Format MapPL sind aufwendige Zeichenregeln zu erstellen, die implizit mit Formatierungsanweisungen im gleichen Dokument zusammenhängen (siehe Anhang A.7). Für Modellierer von Workflowanwendungen soll eine Lösung erarbeitet werden, die einen Zugriff auf die Dienste vereinfacht und dabei die Schnittstellenbeschreibung der Dienste einhält.
- **Generische Objekte und Attribute**: Die Nexus Plattform arbeitet mit generischen Objekten und Attributen. Zurückgegebene Objekte können gleichzeitig mehrere Typen haben und auch mehrere gleichnamige Attribute. Zudem wird durch die Verwendung abstrakter Elemente im AWML Format der ausgetauschte Datentyp nicht genau spezifiziert (siehe Anhang A.4). In den Workflows muss das Problem der Verarbeitung dieser generischen Objekte aufgegriffen werden.

4.1.2 Transportprotokolle und Verteilung in Nexus

Für den Versand von Nachrichten an die Dienste der Nexus Plattform werden verschiedene Transportprotokolle angeboten. Abbildung 4.1 zeigt die verschiedenen Protokolle, die zur Kommunikation mit den jeweiligen Diensten genutzt werden können. Anfragen für den *Spatial Model Server* (Kontextserver), die Föderation und den Kartendienst (Map Service) werden vom

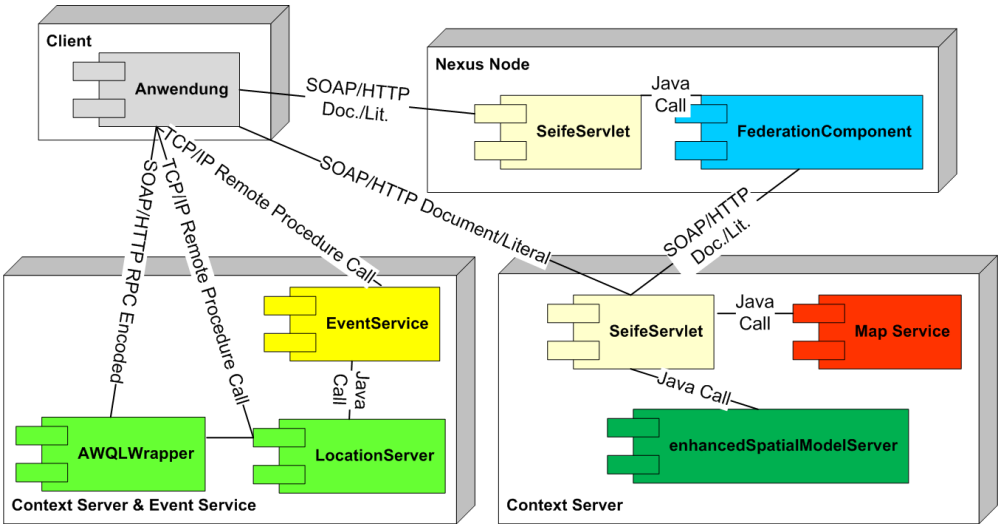

Abbildung 4.1: Momentane Schnittstellen der Nexus Dienste

SeifeServlet als SOAP Nachrichten über HTTP entgegengenommen. Dabei wird als Nachrichtenstil Document/Literal verwendet, d.h. die SOAP Nachricht enthält im Body ein Dokument, welches durch XML Schema beschrieben ist (siehe Abschnitt 2.3.2). Das SeifeServlet erkennt anhand des Dokumenttyps, welchen Service es aufrufen muss. Das SeifeServlet erwartet für die Dienste Föderation und Kontextserver Dokumente vom Typ AWQL und AWML (siehe Abschnitt 4.1.1) und für den Kartendienst Dokumente vom Format MapPL. Da der *Location Service* (Kontextserver) ursprünglich ein eigenständiger Dienst zur Verfolgung von Objektbewegungen war (siehe Abschnitt 2.1.3), bietet er nachträglich zur Erfüllung der Schnittstelle eines Kontextservers für seine *Location Server* einen *Wrapper* an (AWQLWrapper). Der *Wrapper* wandelt AWQL Anfragen in Methodenaufrufe des *Location Servers* um. Die Nachrichten für den Wrapper werden über SOAP/HTTP im RPC/Encoded Stil gesendet. Der Location Server bietet natürlich auch die ursprüngliche Möglichkeit von RPC-Aufrufen über UDP.
Der Ereignisdienst (Event Service) verarbeitet Anfragen über *Remote Procedure Calls* (RPC). Dabei werden sog. Stubs als Stellvertreter-Objekte mit den zur Verfügung stehenden Methoden des Ereignisdiensts von der Anwendung erzeugt. Der Versand der serialisierten Daten erfolgt über TCP/IP. Dabei werden Java Objekte an Stelle von XML Dokumenten versendet, die jeweils die Daten für die Registrierung der Ereignisse enthalten. Der asynchrone Callback zum Versand der Benachrichtigung über eingetretene Ereignisse wird in Java mit Interfaces realisiert (siehe auch [Krü06]). Die Anwendung implementiert ein Interface mit den Methoden, die vom Ereignisdienst beim Eintreten eines Ereignisses aufgerufen werden. Zusätzlich hinterlegt die Anwendung ein *Callback Handle*. Das *Callback Handle* enthält die nötigen Informationen für den Rückruf durch den Ereignisdienst, wie das Transportprotokoll und die transportspezifischen Parameter. Als Transportprotokolle für den Callback werden TCP/IP und SOAP/HTTP zur Verfügung gestellt.

Probleme

- **Unterschiedliche Transportprotokolle**: Der Zugriff auf die Komponenten der Nexus Plattform erfolgt mit unterschiedlichen Protokollen, z.B. UDP, TCP oder HTTP. Für Workflowanwendungen soll hier eine einheitliche Schnittstelle zur Verfügung gestellt werden.

- **Unterschiedliche Objektformate**: Zum Anderen erfolgt der Versand von Daten in unterschiedlichen Formaten, z.B. bei der Föderation im XML Format und beim Ereignisdienst in Form von serialisierten Java Objekten. Für Workflowumgebungen soll aus Gründen der Interoperabilität XML als einheitliches Austauschformat verwendet werden.
- **Unterschiedliche Nachrichtenstile**: Wenn der Versand von XML Daten erfolgt, so werden dabei unterschiedliche Stile implementiert, z.B. RPC/Encoded oder Document/Literal. Im Fall des AWQLWrappers wird beispielsweise aufgrund des RPC/Encoded Stils eine AWQL Anfrage im String Format gesendet, wodurch die Struktur der Nachricht verloren geht. Außerdem muss durch den RPC-Charakter zusätzlich die Methode, die aufgerufen werden soll, in der SOAP Nachricht angegeben werden. Im Document/Literal Stil bleibt die Struktur erhalten und die aufzurufende Methode wird anhand des Dokumentformats ermittelt. Für Workflowumgebungen soll deshalb Document/Literal als einheitlicher Stil verwendet werden.

4.1.3 Funktionale und nichtfunktionale Anforderungen

Neben den zuvor genannten Problemen, müssen beim Entwurf des Systems zusätzlich die folgenden funktionalen Anforderungen beachtet werden:

- Die Nexus Plattform unterstützt die Beobachtung von Ereignissen. Ereignisse werden dabei asynchron verarbeitet, da nicht im Voraus bekannt ist, wann ein Ereignis eintrifft. Im Workflow muss diese asynchrone Verarbeitung ebenfalls aufgegriffen werden.
- Sämtliche Schnittstellen sollen vollständig sein, d.h. es sollte jede Funktion, die von den einzelnen Nexusdiensten nach außen hin angeboten wird, auch in der Workflowumgebung verfügbar sein.
- Eine Operation sollte in der Workflowumgebung dasselbe Ergebnis liefern, wie außerhalb der Workflowumgebung. Eine AWQL Anfrage sollte beispielsweise das gleiche Ergebnis liefern wie in bisherigen klassischen Anwendungen.
- Die Schnittstellen, die von der Workflowumgebung verwendet werden, sollen das Format der Nexusdienste einhalten. Es sollte die Spezifikation von Formaten wie beispielsweise AWQL eingehalten werden.
- Die Datentypen, die in der Workflowumgebung genutzt werden und mit den Nexusdiensten ausgetauscht werden, sollten möglichst das spezifizierte Format der jeweiligen Nexus Dienste unterstützen (z.B. durch Verwendung der Datentypen aus NSAT oder NSAS aus Anhang A.1).
- Der Zugriff auf einen bestimmten Dienst der Nexus Plattform soll in der Workflowumgebung erkenntlich sein, d.h. es muss klar sein, dass beispielsweise der Ereignisdienst aufgerufen wird.
- Änderungen an den Nexusdiensten sollten möglichst keine Änderungen an der Workflowumgebung nach sich ziehen oder möglichst geringe Änderungen erfordern (z.B. nach der Erweiterung durch neue Funktionen oder nach der Änderung von Parametern).

Außerdem gibt es noch einige nichtfunktionale Anforderung, die bei einer Integration der Nexusdienste in eine Workflowumgebung zu beachten sind:

- Die Verwendung der Nexusdienste im Workflow sollte einfach gehalten sein, d.h. der Programmierer sollte nach einer kurzen Einlernphase in gewohnter Weise seine Arbeit

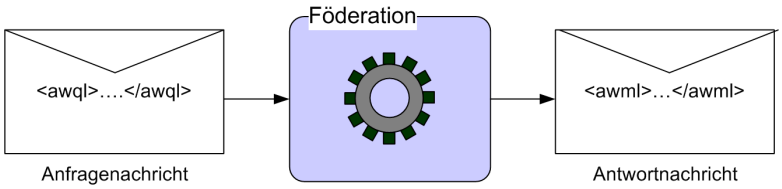

Abbildung 4.2: Verarbeitung komplexer Nachrichten in der Föderation

durchführen können.

- Die Bearbeitungszeit der Anfragen sollte sich in der Workflowumgebung nicht deutlich verschlechtern bzw. falls möglich verbessern.
- Anwendungen einer Workflowumgebung sollten im Vergleich zur klassischen Programmierung möglichst den gleichen oder einen geringeren Programmieraufwand aufweisen, soweit sie vergleichbar sind.

Bei der Durchführung der Arbeit sollten diese Anforderungen beachtet werden und das Ergebnis daraufhin evaluiert werden.

4.2 Grobentwurf - Konzept und Architektur

In diesem Abschnitt wird aufbauend auf der Anforderungsanalyse aus Abschnitt 4.1 der Grobentwurf für die Integration von Diensten der Nexus Plattform in Workflowumgebungen erstellt. Dabei sollen die Ziele zur Lösung der Probleme und Anforderungen auf hoher Ebene formuliert werden. Im Feinentwurf (siehe Abschnitt 4.3) werden diese Ziele anschließend ausgearbeitet. Für die Integration der Nexusdienste sind folgende Erweiterungen notwendig:

- Entwicklung von elf Integrationsprozessen für die Integration der Nexusdienste in Workflows:
  - Modellierung der Prozesse mit BPEL
  - Beschreibung der Prozess-Schnittstellen mit WSDL
  - Definition einfacher Nachrichtenformate für die Prozesse mit XML
  - Erstellung von Transformationsvorschriften für die Nachrichten mit XSLT
- Implementierung einer zusätzlichen Web Service Schnittstelle für den Ereignisdienst
- Beschreibung der Web Service Schnittstellen der Nexusdienste mit WSDL
- Realisierung einer asynchronen Ereignisverarbeitung mit WS-Addressing

4.2.1 Basiskonzept: Transformation und hierarchische *Integrationsprozesse*

Nachrichtentransformation: Für die Integration von Workflows in die Nexus Plattform ist die Handhabung der verschiedenen und zum Teil komplexen Nachrichtenformate von Nexus wesentlich. Die Nachrichtenformate sind sehr komplex, da sie mehrere Funktionalitäten in sich vereinen oder generische Objekte definieren. In Listing A.2 wurde dazu das Schema des Nachrichtenformats für die Abfrage von Modelldaten der Augmented World aufgeführt, das AWQL

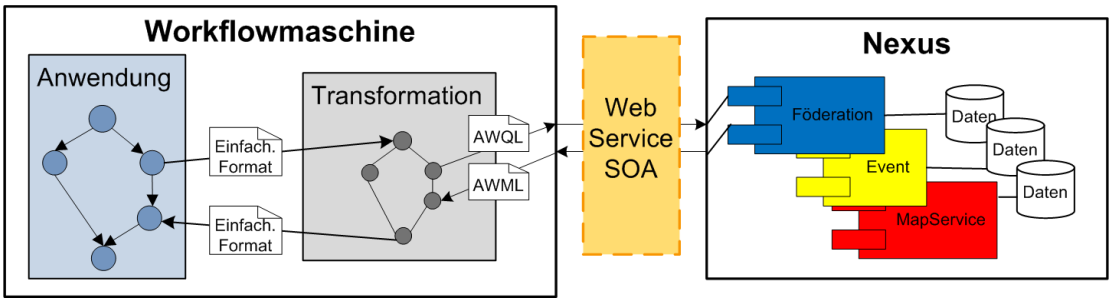

Abbildung 4.3: Integration von Nexusdiensten durch Transformation von Nachrichtenformaten

Format. In Listing A.3 wurde zudem das Format für die Rückgabe der ermittelten Objekte beschrieben, das AWML Format. Wenn beispielsweise ein Aufruf an die Föderation erfolgt, so muss die Anfrage syntaktisch korrekt sein, d.h. es muss ein Dokument im AWQL Format übergeben werden (siehe Abb. 4.2). Zu diesem Zweck ist zum einen eine Einbindung der komplexen Nachrichtenformate in die Workflowumgebung nötig, um die vollständige Funktionalität der Nexus Plattform zu gewährleisten und zum anderen soll die Bereitstellung einfacherer Nachrichtenformate unterstützt werden, die dann in die komplexeren Nexus Nachrichtenformate umgewandelt werden. Diese Aufgaben übernehmen Prozesse, die zwischen der Workflowanwendung und den Nexusdiensten liegen (siehe Abb. 4.3). Diese Prozesse, die sowohl die Einbindung als auch die Transformation von Nachrichten vornehmen, werden im Folgenden als *Integrationsprozesse* bezeichnet. Ein *Integrationsprozess* verwendet für die Umwandlung der Nachrichtenformate XSL Technologien (siehe Abschnitt 2.2.2). Diese Vorgehensweise hat die folgenden Vorteile:

a) In vielen Workflowmaschinen sind bereits XSLT-Prozessoren integriert.

b) Mit einem XSL Stylesheet kann mittels XPath und XSLT ein XML Dokument in verschiedene andere Dokumente transformiert werden, z.B. von einem einfachen Format in ein komplexeres.

c) Transformationsvorschriften (Stylesheets) können schnell ausgetauscht werden. Beim Tausch einer Transformationsvorschrift gegen eine andere muss nur beachtet werden, dass die Schnittstelle zum jeweiligen Nexusdienst eingehalten wird.

d) Aufgrund b) und c) können auch Attribute verarbeitet werden, deren Typ der Workflowmaschine beim Eintreffen des Antwort-Dokuments noch unbekannt ist (s.o. abstrakter Typ). Der Programmierer muss lediglich wissen, welche Attribute zu erwarten sind und kann hierfür dann eine besondere Transformationsvorschrift erzeugen.

Hierarchischer Aufbau: Die *Integrationsprozesse* sind dabei hierarchisch aufgebaut (siehe Abb. 4.4). Auf der ersten Ebene befinden sich fünf Prozesse für die wesentlichen Funktionen, die von der Nexus Plattform zur Verfügung gestellt werden. Über den *ContextQuery Integrationsprozess* kann eine Abfrage von Daten erfolgen. Mit *ContextModeling* können neue Objekte in der Augmented World modelliert werden. Mit *ContextManipulation* können vorhandene Objekte in der Augmented World verändert werden. Mit *GetMap* können Gebietskarten erstellt werden. Schließlich ermöglicht *ContextEvent* die Beobachtung von Ereignissen auf der Nexus Plattform. Die *Integrationsprozesse* auf der ersten Ebene sorgen für eine direkte Einbindung der komplexen Nexus Nachrichtenformate (siehe Abschnitt 4.1.1) und Nexusdienste in die Workflowumgebung. Auf der zweiten Ebene befinden sich dann *Integrationsprozesse*, die basie-

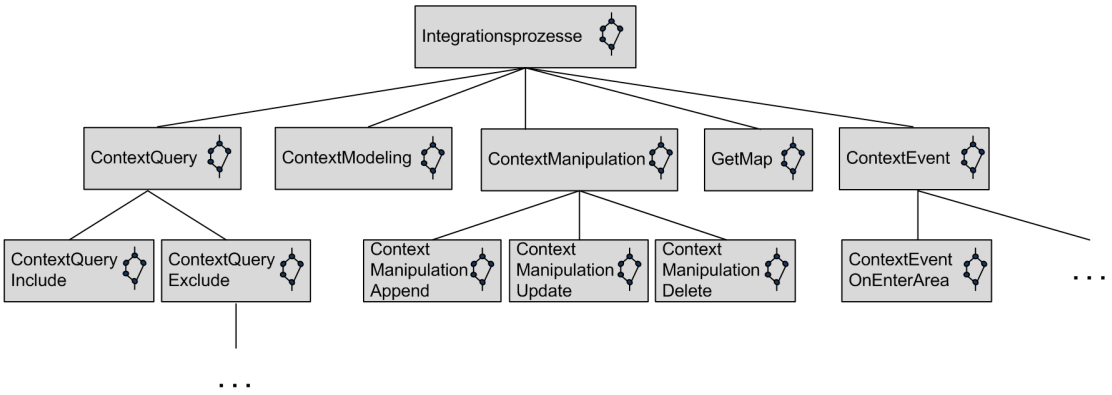

Abbildung 4.4: *Integrationsprozesse* zur Bereitstellung von Kontextdaten in Workflows

rend auf den Prozessen der ersten Ebene eine einfache Schnittstelle zur Verfügung stellen und deshalb eine Transformation von Nachrichtenformaten vornehmen. Hier werden beispielhaft sechs weitere *Integrationsprozesse* entworfen. *ContextQueryInclude* und *ContextQueryExclude* vereinfachen den *Integrationsprozess ContextQuery*, indem die Funktionalität zum Ein- bzw. Ausblenden von Objektattributen bei *ContextQuery* durch zwei separate Prozesse unterschieden wird. *ContextManipulationAppend*, *ContextManipulationUpdate* und *ContextManipulationDelete* vereinfachen den *Integrationsprozess ContextManipulation*, indem sie die Funktionen zum Einfügen bzw. Aktualisieren von Objektattributen und zum Löschen von Objekten als eigene Prozesse bereitstellen. *ContextEventOnEnterArea* vereinfacht den *Integrationsprozess ContextEvent*, indem er die Registrierung von Ereignissen auf *On Enter Area* Ereignisse beschränkt. Die *Integrationsprozesse* lassen sich ab der zweiten Ebene sowohl vertikal als auch horizontal erweitern. **Diese Erweiterungsmöglichkeit soll die wesentliche Eigenschaft von Nexus aufgreifen: Als Plattform für unterschiedliche Anwendungsbereiche zu dienen**. So kann beispielsweise *ContextQueryExclude* auf der dritten Ebene verfeinert werden, um die Ausblendung bestimmter Attribute dauerhaft zu erreichen, z.B. die Position eines Objekts soll aus Datenschutzgründen stets ausgeblendet werden. Horizontal können auf der zweiten Ebene beispielsweise zusätzliche *Integrationsprozesse* für das Ereignis *On Meeting* erstellt werden oder ein *Integrationsprozess ContextModelingWorker* für die Modellierung eines Arbeiters in einer Unternehmensanwendung. Die *Integrationsprozesse* können folglich auf Basis dieses Konzepts anwendungsspezifisch erweitert werden.
Generische Objekte müssen ebenfalls für die Workflowanwendung aufbereitet werden. Die vorgeschlagenen Integrationsprozesse leiten dafür die anwendungsspezifischen Rückgaben der Nexus Plattform direkt an die Anwendung weiter. Für diese anwendungsspezifischen Rückgaben können für unterschiedliche Anwendungsbereiche *Integrationsprozesse* auf Basis der hier erarbeiteten Konzepte erstellt werden. Im Abschnitt 4.6 sehen wir dies an einer beispielhaften Workflowanwendung, in der die abstrakten Elemente aus einer AWML Nachricht durch anwendungsspezifische Umwandlung verarbeitet werden.

4.2.2 Entwicklung einer serviceorientierten Architektur für die Integration von Nexusdiensten

In Abbildung 4.5 sehen wir die erarbeitete Architektur, die eine Einbindung der Nexus Plattform in eine Workflowumgebung ermöglicht. Die Architektur basiert auf Elementen der Web Service

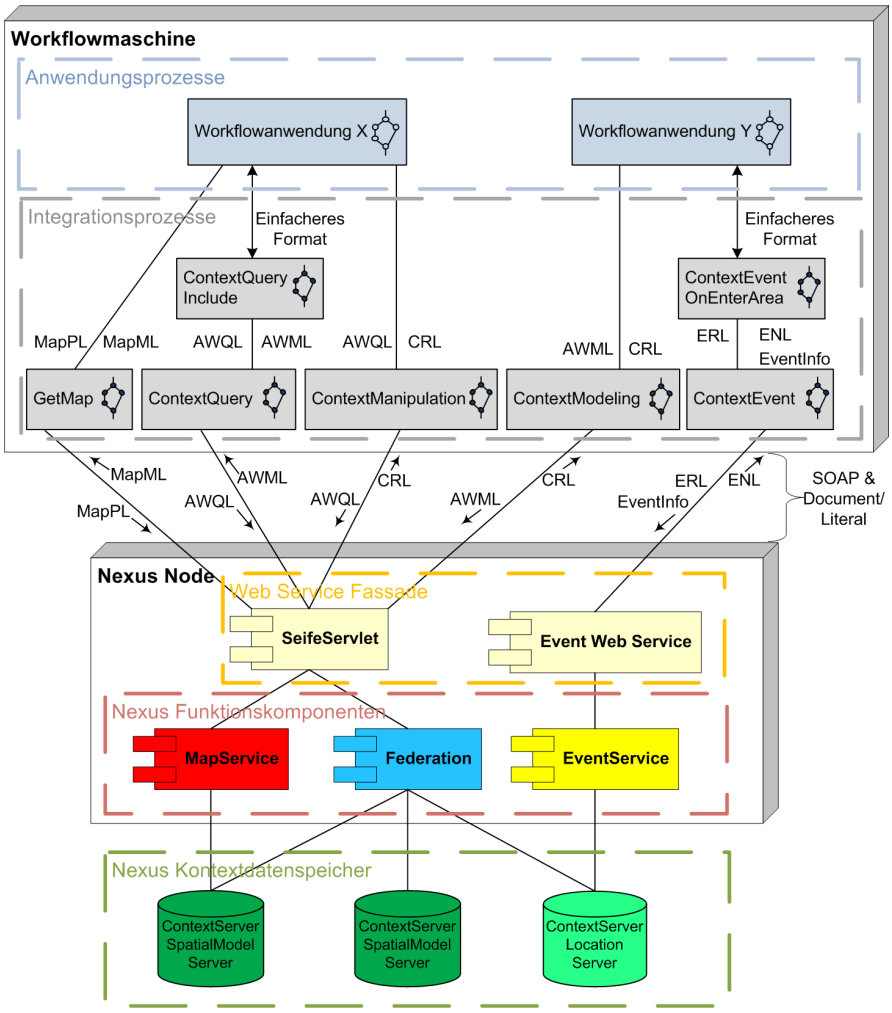

Abbildung 4.5: Architektur für die Integration von Nexusdiensten in Workflows

Architektur. Hierfür ist die Erstellung zusätzlicher Web Service Schnittstellen und die Beschreibung der Dienste erforderlich (Web Service Fassade). Damit eine Interoperabilität zwischen verschiedenen Implementierungen von Workflowmaschinen erzielt werden kann, empfiehlt sich die Realisierung des Nachrichtentransports über den SOAP Standard im Document/Literal Nachrichtenstil. Die WS-SOA ermöglicht das Einbinden von Erweiterungen wie Sicherheit und Transaktion auch im Nachhinein, wie z.B. über die Definition von zusätzlichen Header Elementen innerhalb von SOAP Nachrichten. Die Implementierung der Web Service Schnittstelle kann durch existierende Frameworks unterstützt werden (siehe Abschnitt 2.3.2).

4.2.3 Asynchrone Ereignisverarbeitung mit WS-Addressing

Die asynchrone Verarbeitung des Ereignisdiensts wird in Workflows durch WS-Addressing Header (siehe Abschnitt 2.3) aufgegriffen. In einer Anfrage an den Ereignisdienst werden Informationen für einen nachträglichen Rückruf hinterlegt. In der Zwischenzeit wird die aktuelle Prozessaktivität „konserviert". Sobald der Ereignisdienst ein Ereignis bemerkt, verwendet er die Informationen aus dem WS-Addressing Header für den Versand der Antwortnachricht. Die

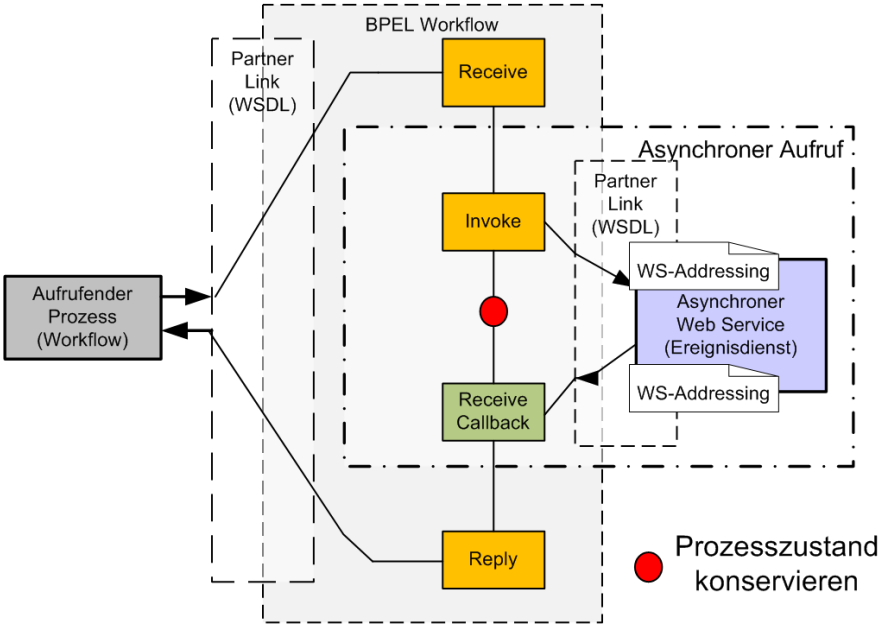

Abbildung 4.6: Aufruf eines asynchronen Nexusdiensts in einem Workflow

Workflowmaschine kümmert sich um das Auffinden des „konservierten" Prozesses. Für diese Form der Zuordnung muss die Web Service Schnittstelle des Ereignisdiensts (siehe Abb. 4.5) die WS-Addressing Header verarbeiten können. Abbildung 4.6 veranschaulicht diesen Vorgang. Der asynchrone Aufruf ist dabei hervorgehoben. Der rote Punkt symbolisiert das Festhalten des Prozesszustandes in der Workflowmaschine, während der asynchrone Service seine Verarbeitung durchführt. Zu diesem Zweck muss für den Ereignisdienst der Nexus Plattform eine Web Service Schnittstelle erzeugt werden, in der die WS-Addressing Spezifikation implementiert ist. Die Verwendung von WS-Addressing erfordert keine zusätzliche Anpassung der ausgetauschten Nachrichtenformate und die Ereignisverarbeitung bleibt für den Modellierer eines Workflows transparent.

4.3 Feinentwurf - Vorstellung der Integrationsprozesse

In diesem Teil der Arbeit sollen die Konzepte aus Abschnitt 4.2 verfeinert werden. Der Aufbau dieses Kapitels orientiert sich dabei an den Integrationsprozessen.
Im Abschnitt *ContextQuery* wird der Integrationsprozess für die Abfrage von Kontextdaten verfeinert und es werden die Unterschiede von *ContextQueryInclude* und *ContextQueryExclude* herausgearbeitet. Mit *ContextManipulation* wird der Integrationsprozess zur Veränderung bereits vorhandener Datenbestände eingeführt. Die Integrationsprozesse *ContextManipulationAppend*, *ContextManipulationUpdate* und *ContextManipulationDelete* vereinfachen anschließend den Integrationsprozess *ContextManipulation*. Der Integrationsprozess *ContextModeling* wird für die Modellierung von neuen Daten in der Augmented World verwendet. Zur Registrierung neuer Ereignisse wird *ContextEvent* eingeführt. In diesem Zusammenhang wird auch ein einfacherer Integrationsprozess vorgestellt, der speziell für das Ereignis *On Enter Area* entworfen wird. Zum Schluss wird mit *GetMap* ein Integrationsprozess für die Funktionen des Kartendiensts präsentiert.

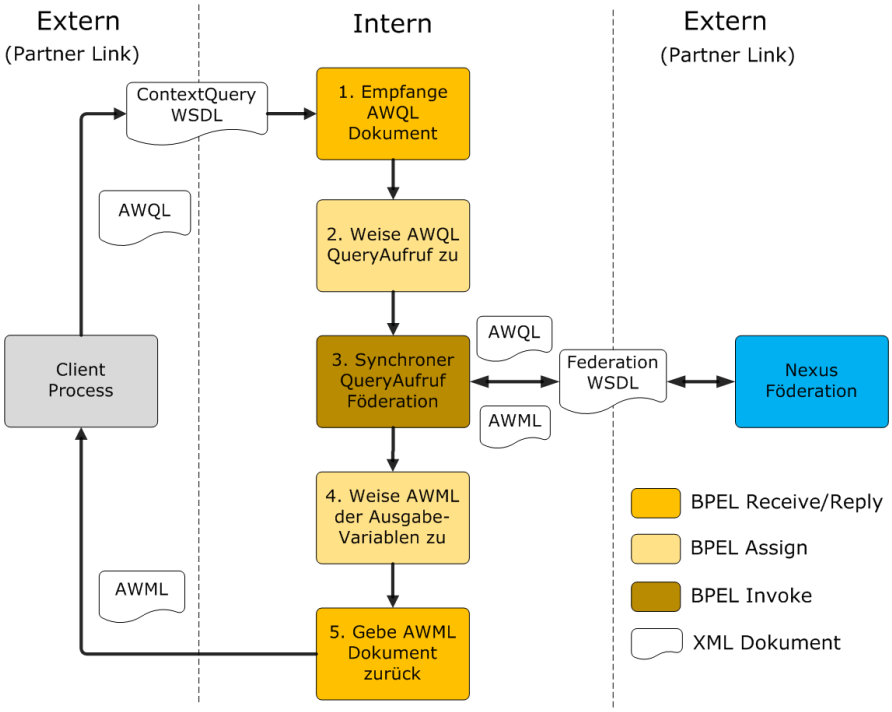

Abbildung 4.7: Prozessmodell von *ContextQuery*

Für jeden Integrationsprozess wird die entsprechende Web Service Schnittstelle für den Zugriff auf die entsprechenden Nexusdienste beschrieben. Außerdem wird für jeden Integrationsprozess ein Prozessmodell erstellt, das den Zugriff auf die Nexusdienste modelliert. Zudem werden die Transformationen in den Integrationsprozessen verfeinert, die eine Umwandlung der Nachrichtenformate ermöglichen. Für ein besseres Verständnis der Integrationsprozesse werden schließlich Beispielanfragen aufgeführt.

4.3.1 ContextQuery

Mit *ContextQuery* können Daten aus der Augmented World abgefragt werden. Bei einer Anfrage über diesen Integrationsprozess werden Daten an die Föderation im AWQL Format gesendet. Das Resultat der Anfrage wird im AWML Format erwartet und an den Aufrufer weitergeleitet.

Prozessmodellierung

In Abbildung 4.7 wird das Prozessmodell für ContextQuery dargestellt. Der Prozess stellt seinen Dienst über eine WSDL Beschreibung nach außen hin zur Verfügung (siehe Abschnitt 2.3.2). Dabei erwartet ContextQuery als Eingabe ein Dokument im AWQL Format. Die Eingabe wird dann dem Aufruf an die Föderation als Parameter zugewiesen. Die Dienste der Föderation werden ebenfalls über eine WSDL-Beschreibung in Anspruch genommen (s.u.). Der Aufruf der Föderation läuft synchron ab, d.h. der Aufrufer ist solange blockiert bis eine Antwort von der Föderation geliefert wird. Die Nachricht, die als Resultat von der Föderation geliefert wird, ist im AWML Format. Die AWML Nachricht wird vom ContextQuery Prozess direkt an den Aufrufer zurückgegeben.

Listing 4.1: Beispielanfrage ContextQuery

```
<awql
    xmlns="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
    xmlns:nsas="http://www.nexus.uni-stuttgart.de/1.0/NSAS">
  <geoDataFormat>WKT</geoDataFormat>
  <restriction>
    <and>
        <equal>
            <target>nsas:name.nsas:value</target>
            <referenceValue>Lagerhalle XY</referenceValue>
        </equal>
    </and>
  </restriction>
</awql>
```

Ausgetauschte Datentypen

Es werden sowohl zwischen dem aufrufenden Klienten-Prozess, als auch zwischen ContextQuery und der Föderation, Nachrichten im Format AWQL und AWML ausgetauscht. Die Schnittstellen sind also identisch. Für die Beschreibung der Datentypen, die dem AWQL und dem AWML Format entsprechen, werden bereits vorhandene XML Schemata verwendet (siehe Anhang A.4).

Web Service Schnittstelle

Für die Verarbeitung auf der Nexus Plattform existiert bereits eine Service Implementierung (siehe Abschnitt 4.4.1), welche Anfragen im AWQL Format entgegennimmt und Resultate im AWML Format zurücksendet. Für diesen Dienst wird deshalb eine WSDL-Beschreibung erzeugt, die für die Föderation den Nachrichtenstil Document/Literal spezifiziert. Für die Anfrage durch den Prozess ContextQuery wird in der WSDL-Beschreibung eine Operation *queryContextData* spezifiziert. Diese Operation erwartet zum Abruf von Modelldaten ein XML Dokument, das dem AWQL Schema entspricht. Die Rückgabenachricht der Operation ist ein XML Dokument, das dem AWML Schema entspricht. Die vorhandenen Schema Dateien werden deshalb im types Bereich des WSDL Dokuments importiert und für die Deklaration der Nachrichten verwendet. Die Dokumente im AWQL- und AWML-Format werden über SOAP/HTTP versendet, weshalb für die queryContextData Operation ein SOAP-Binding deklariert wird. Im service Element der WSDL-Beschreibung ist dann noch die Adresse der Service Implementierung eingetragen. Für Aufrufe durch BPEL Prozesse wird in der WSDL-Beschreibung ein partnerLink (siehe Abschnitt 2.4.2) erzeugt, in dem die Föderation die Rolle des Dienstanbieters (FederationServiceProvider) erhält.

Transformation

Der Integrationsprozess *ContextQuery* sorgt für die Einbindung der komplexen Nachrichtenformate AWQL und AWML, um die vollständige Funktionalität für die Abfrage von Daten auf der Nexus Plattform bereitzustellen. Er nimmt folglich keine Transformation von Ein- und Ausgabeformaten vor. Eine Transformation erfolgt in den Integrationsprozessen auf der zweiten

Listing 4.2: Beispielantwort ContextQuery

```
<awml
    xmlns="http://www.nexus.uni-stuttgart.de/2.0/AWML"
    xmlns:nsas="http://www.nexus.uni-stuttgart.de/1.0/NSAS"
    xmlns:nscs="http://www.nexus.uni-stuttgart.de/1.0/NSCS">
  <nexusobject>
    <nsas:type>
      <nsas:value>nscs:Building</nsas:value>
    </nsas:type>
    <nsas:nol>
      <nsas:value>(...StaticObjectLocator...)</nsas:value>
    </nsas:nol>
    <nsas:name>
      <nsas:value>Lagerhalle XY</nsas:value>
    </nsas:name>
  </nexusobject>
</awml>
```

Ebene (siehe z.B. *ContextQueryExclude*).

Beispiel für Anfrage- und Antwortnachricht

In Listing 4.1 ist eine Beispielanfrage im AWQL Format aufgeführt, die der ContextQuery Prozess entgegennimmt, und in Listing 4.2 ist das entsprechende Resultat zu sehen, das der ContextQuery Prozess im AWML Format zurückgibt. Bei der Anfrage muss der aufrufende Prozess immer dafür sorgen, dass die Namespaces der XML Schema Dokumente, die er für die Anfrage verwendet, im Dokument angeben werden. Hierfür sollten die entsprechenden Informationen aus Tabelle 4.1 genutzt werden. Die Beispielanfrage sucht nach einem Gebäudeobjekt, das den Namen „Lagerhalle XY" trägt. Das Rückgabeformat für geografische Daten soll dabei WKT sein. Es wird für die Anfrage neben dem Namensraum für das AWQL Schema zusätzlich das NSAS für die Referenzierung des Attributs in der Auswahlbedingung benötigt (siehe Anhang A.1). Die Rückgabe enthält das entsprechende Objekt mit sämtlichen Attributen. Das Objekt gehört der Standardklasse Building an und hat bisher nur einen eindeutigen Bezeichner und einen Namen als zusätzliche Attribute.

ContextQueryExclude: ContextQueryExclude ist ein erweiterter Integrationsprozess zur Abfrage von Daten aus der Augmented World. Mit diesem Prozess ist es möglich, bestimmte Attribute in der Ergebnismenge auszublenden. In Abbildung 4.8 ist das entsprechende Prozessmodell des Integrationsprozesses abgebildet. Im zweiten Schritt wird eine Transformation vorgenommen. Der Grund hierfür ist, dass für die Schnittstelle zum Klienten-Prozess ein vom AWQL abweichendes Austauschformat für Nachrichten gewählt worden ist. Die Elemente *aa*, *ecs*, *srs*, *geoDataFormat*, *nearestNeighbor* und *restriction* aus dem AWQL Format sind auch im ContextQueryExclude Format vorhanden (siehe Listing A.2). Anstatt des komplexen *choice* Elements wird in diesem vereinfachten Format jedoch lediglich das *target* Element vom AWQL Schema verwendet. Das target Element basiert auf dem einfachen XML Schema Typ *string*. Aus Gründen der Einheitlichkeit bei der Wahl der Bezeichner wurde auf das vorhandene target Element aus dem AWQL Schema referenziert. Der Wert, der in diesem Element erwartet wird, ist entsprechend der Definition in AWQL ein Attributtyp in Punktnotation. Dieser Wert gibt an,

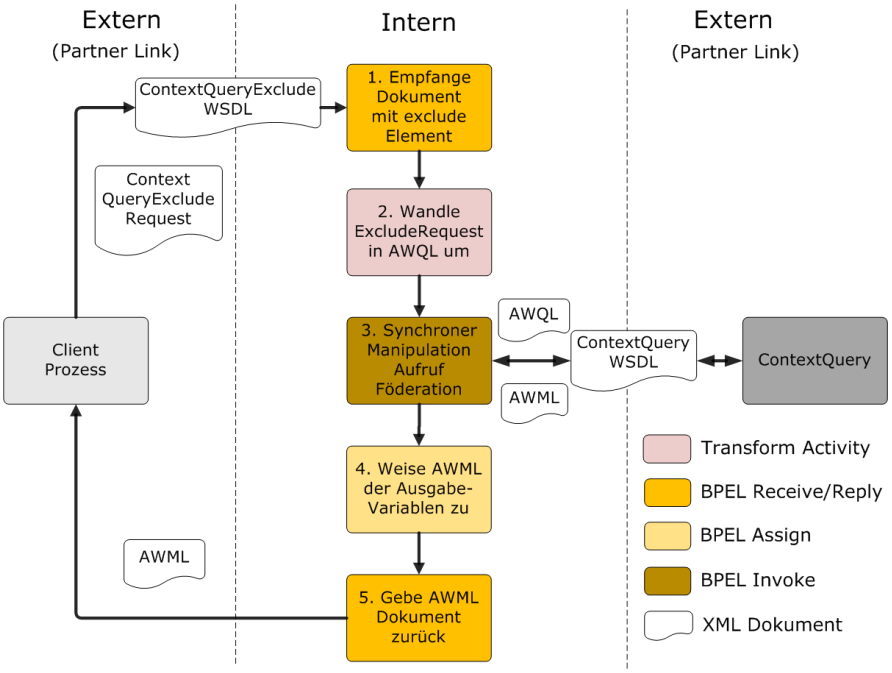

Abbildung 4.8: Prozessmodell von *ContextQueryExclude*

welches Attribut nicht in der Auswahlmenge angezeigt werden soll (siehe Anhang A.5). Das target Element sollte mindestens einmal vorkommen (minOccurs=1), kann aber auch mehrmals vorkommen (maxOccurs=unbounded).
In Listing 4.3 ist eine Beispielanfrage aufgeführt, die vom Klienten-Prozess an die ContextQueryExclude gesendet wird. Die Anfrage ist vom Format ContextQueryExcludeRequest. Das ContextQueryExcludeRequest Format hat dabei einen eigenen Namensraum, um Kollisionen mit Elementen anderer Datenformate zu vermeiden. Mit der Anfrage möchte der Klienten-Prozess das Objekt mit dem Namen „Lagerhalle XY" erhalten. Die Bedingung für die Suche nach dem Objekt ist genauso formuliert wie im Beispiel von ContextQuery. Der aufrufende Klienten-Prozess möchte jetzt mit der Anfrage aber den Wert im *nol* Attribut ausblenden. Werte werden bei Attributen aus dem NSAS im untergeordneten *value* Element gehalten. Deshalb wird im target Element der Ausdruck *nsas:nol.nsas:value* angegeben.
Listing 4.4 zeigt die Rückgabe von ContextQueryExclude an den aufrufenden Klienten-Prozess.

Listing 4.3: Beispielanfrage ContextQueryExclude

```
<ContextQueryExcludeProcessRequest
  xmlns="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQueryExclude"
  xmlns:awql="http://www.nexus.uni-stuttgart.de/2.0/AWQL">
    <awql:restriction>
      <awql:equal>
        <awql:target>nsas:name.nsas:value</awql:target>
        <awql:referenceValue>Lagerhalle XY</awql:referenceValue>
      </awql:equal>
    </awql:restriction>
    <awql:target>nsas:nol.nsas:value</awql:target>
</ContextQueryExcludeProcessRequest>
```

Listing 4.4: Beispielantwort ContextQueryExclude

```
<awml
   xmlns="http://www.nexus.uni-stuttgart.de/2.0/AWML"
   xmlns:nsas="http://www.nexus.uni-stuttgart.de/1.0/NSAS"
   xmlns:nscs="http://www.nexus.uni-stuttgart.de/1.0/NSCS">
     <nexusobject>
       <nsas:type>
         <nsas:value>nscs:Building</nsas:value>
       </nsas:type>
       <nsas:name>
         <nsas:value>Lagerhalle XY</nsas:value>
       </nsas:name>
     </nexusobject>
</awml>
```

Dabei handelt es sich wie beim ContextQuery Integrationsprozess um ein Dokument im AWML Format, das lediglich an den Klienten weitergeleitet wird. Der eindeutige Bezeichner wird im Vergleich zur Rückgabe von ContextQuery (siehe Listing 4.2) nicht angezeigt, da er durch den Prozess ausgeblendet wurde.
Die Transformation der Daten vom ContextQueryExcludeRequest Format in das AWQL Format geschieht mit einem XSLT Stylesheet (siehe Abschnitt 2.2.2). In Abbildung 4.9 ist die Transformation der Dokumente bildhaft dargestellt. Dabei wird aus den Eingabedaten, die der Klient der ContextQueryExclude Schnittstelle übergibt, über das Stylesheet ein gültiges AWQL Dokument erzeugt. Das Stylesheet fügt zusätzlich die Namensräume der Standardschemata von Nexus ein, die für die Bearbeitung der Anfrage nötig sind. Dieses AWQL Dokument wird dann für die weitere Bearbeitung an den ContextQuery Integrationsprozess gesendet. Der Aufruf von ContextQuery ist hier ebenfalls synchron, das bedeutet der ContextQueryExclude Prozess bleibt bis zur Antwort blockiert.

ContextQueryInclude: Der ContextQueryInclude Integrationsprozess hat die umgekehrte Funktionalität zu ContextQueryExclude. Der Klienten-Prozess kann über eine Anfrage an diese Schnittstelle die Attribute angeben, die bei den Objekten der Auswahlmenge angezeigt werden sollen. Alle Attribute, die nicht ausdrücklich angegeben werden, werden im Resultat auch nicht angezeigt.

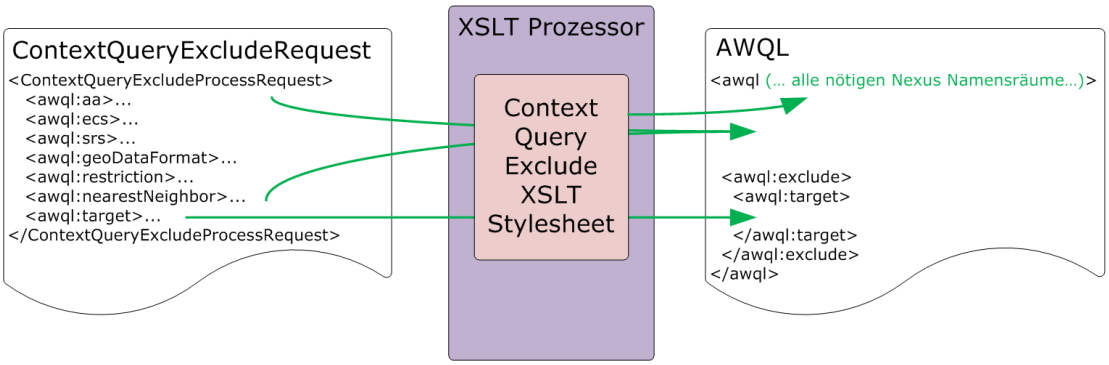

Abbildung 4.9: Transformation der Dokumente bei *ContextQueryExclude*

Listing 4.5: Struktur des include Elements aus dem AWQL Schema

```
<element name="include">
  <complexType>
    <sequence>
      <element ref="awql:target" minOccurs="0"/>
      <element name="include" minOccurs="0">
        <complexType>
          <sequence>
            <element ref="awql:target" minOccurs="0" maxOccurs="unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element ref="awql:restriction" minOccurs="0"/>
    </sequence>
  </complexType>
</element>
```

Für den ContextQueryInclude Integrationsprozess wird ebenfalls ein eigenes Dokumentformat festgelegt, das *ContextQueryIncludeRequest* Format. Hier werden ebenfalls die Elemente *aa*, *ecs*, *srs*, *geoDataFormat*, *nearestNeighbor* und *restriction* aus dem AWQL Format definiert. Anstelle des komplexen *choice* Elements wird hier aber das einfachere *include* Element aus dem AWQL Schema verwendet. Die Struktur des include Elements ist in Listing 4.5 aufgeführt. Das Element besteht aus drei Teilen, von denen lediglich das target Element obligatorisch ist:

- Das **target** Element ist bereits von der Schnittstelle ContextQueryExclude bekannt. Hier kann das Attribut in Punktnotation angegeben werden, welches angezeigt werden soll.
- Ein weiterer Bestandteil ist ein **untergeordnetes include** Element. Das Element trägt zwar auch die Bezeichnung *include*, ist aber nur innerhalb des übergeordneten include Elements gültig. Mit diesem Element wird die Auswahl der Attributtypen genauer spezifiziert. Beispiel: Eine Gebäude hat als Attribut eine *Adresse*. Die *Adresse* wiederum besteht aus mehreren Elementen, wie z.B. *Straße*, *Hausnummer*, *Ort*. Wird im target Element das Attribut *Adresse* gewählt, so kann im untergeordneten include Element nun der *Ort* gewählt werden. Somit wird auch nur der Ort in der resultierenden Objektmenge angezeigt.
- Ein zusätzlicher Bestandteil des include Elements ist das bereits bekannte **restriction** Element zur Auswahl von Objekten. Mit einer Bedingung innerhalb des include Elements können aus der Gesamtmenge die Objekte gewählt werden, bei denen das include Element Wirkung zeigen soll. Beispiel: Das Attribut Ort soll nur für diejenigen Objekte der gesamten Auswahlmenge angezeigt werden, die sich in Stuttgart befinden.

Es können ein bis mehrere Attribute vom Klienten-Prozess ausgewählt werden, weshalb das include Element auch ein- bis mehrmals vorkommen kann. Für die Umwandlung der Nachrichtenformate ist ein Transformationsschritt nötig. Dabei läuft die Umwandlung ähnlich ab wie in Abbildung 4.9, wobei aus dem ContextQueryIncludeRequest das komplette include Element in das AWQL Dokument übernommen wird. Das Stylesheet sorgt ebenfalls dafür, dass alle nötigen Nexus Namensräume im AWQL Dokument eingefügt werden. Der ContextQueryInclude Integrationsprozess hätte ähnlich ContextQueryExclude auch ein einfacheres Format mit einem einzigen target Element definieren können. Die Wahl des Nachrichtenformats hier zeigt nun deutlich die Möglichkeiten einer beliebigen Erweiterungsmöglichkeit der Integrationsprozesse auf der zweiten Ebene.

Listing 4.6: Beispielanfrage ContextQueryInclude

```
<ContextQueryIncludeProcessRequest
  xmlns="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQueryInclude"
  xmlns:awql="http://www.nexus.uni-stuttgart.de/2.0/AWQL">
    <awql:restriction>
      <awql:equal>
        <awql:target>nsas:name.nsas:value</awql:target>
        <awql:referenceValue>Lagerhalle XY</awql:referenceValue>
      </awql:equal>
    </awql:restriction>
    <awql:include>
      <awql:target>nsas:type.nsas:value</awql:target>
    </awql:include>
    <awql:include>
      <awql:target>nsas:name.nsas:value</awql:target>
    </awql:include>
    <awql:include>
      <awql:target>nsas:address</awql:target>
        <awql:include>
          <awql:target>nsas:value.nsas:city</awql:target>
        </awql:include>
    </awql:include>
</ContextQueryIncludeProcessRequest>
```

In Listing 4.6 sehen wir eine Beispielanfrage an den ContextQueryInclude Prozess. Dabei wird nach einem Objekt gesucht, das die Bezeichnung Lagerhalle XY trägt. Diesmal sollen drei ausgewählte Attribute angezeigt werden. Angezeigt werden sollen der Name, der Typ des Objekts und der Standort. Deshalb werden drei include Elemente benötigt. Dabei ist der Standort ein Teil der Adresse des Objekts, weshalb bei einem include Element ein untergeordnetes include verwendet wird, um nur den Ort auswählen zu können. Würde der Klienten-Prozess die gesamte Adresse benötigen, so hätte er bereits im übergeordneten include Element den Wert *nsas:address.nsas:value* eingetragen und das untergeordnete include Element weggelassen. Das entsprechende Ergebnis der Anfrage ist in Listing 4.7 aufgeführt. Die Namensräume wurden dabei zur besseren Übersicht im Beispiel entfernt.

4.3.2 ContextManipulation

Bislang wurden Integrationsprozesse präsentiert, bei denen eine Abfrage von Objekten erfolgte. Wenn Objekte der Augmented World verändert werden sollen, wird der *ContextManipulation* Integrationsprozess benötigt. Dieser Prozess ermöglicht, zusammen mit den Prozessen *ContextManipulationAppend*, *ContextManipulationUpdate* und *ContextManipulationDelete*, die Veränderung bereits vorhandener Objekte.

Prozessmodellierung

In Abbildung 4.10 ist der Prozess von ContextManipulation dargestellt. Erwartet wird ein Dokument im Format ContextManipulationRequest (siehe Datentypen). Es werden ebenfalls die Dienste der Föderation mit einem synchronen Aufruf in Anspruch genommen (siehe Web Service Schnittstelle). Die Eingabe wird vor dem Aufruf an die Föderation in ein AWQL Dokument umgewandelt. Die Föderation liefert ein Resultat im CRL Format, welches direkt an den Aufrufer weitergeleitet wird.

Listing 4.7: Beispielantwort ContextQueryInclude

```
<awml>
  <nexusobject>
    <type>
      <value>nscs:Building</value>
    </type>
    <name>
      <value>Lagerhalle XY</value>
    </name>
    <address>
      <value>
        <city>Stuttgart</city>
      </value>
    </address>
  </nexusobject>
</awml>
```

Ausgetauschte Datentypen

ContextManipulation erwartet vom Klienten-Prozess ein Dokument im Format ContextManipulationRequest. Neben den Elementen *aa*, *ecs*, *nearestNeighbor* und *restriction* aus dem AWQL Format wird zusätzlich das Element *modifyingElement* definiert. Das modifyingElement ist vom XML Schema Typ *anyType* und erwartet ein update, append oder delete Element aus dem AWQL Schema. ContextManipulation bündelt die Operationen zur Veränderung von existenten Daten im modifyingElement. In der praktischen Anwendung wird ContextManipulation kaum Anwendung finden und einem der drei spezielleren Integrationsprozesse *ContextManipulationAppend*, *ContextManipulationUpdate*, *ContextManipulationDelete* (s.u.) weichen müssen. ContextManipulation wurde aus Konsistenzgründen dennoch entworfen. Änderungen am Augmented World Model werden mit einem AWQL Dokument durchgeführt und von der Föderation mit einem CRL Dokument bestätigt. Das CRL Dokument wird an den aufrufenden Klient weitergeleitet.

Web Service Schnittstelle

Die ContextManipulation Schnittstelle verwendet für Änderungsoperationen ebenfalls die Schnittstelle der Föderation, die bereits in Abschnitt 4.3.1 vorgestellt wurde. Dabei wird für Änderungsoperationen in der WSDL-Beschreibung eine Operation *modifyContextData* spezifiziert. Die Operation erwartet zur Änderung von Modelldaten ein XML Dokument, das dem AWQL Schema entspricht. Die Rückgabenachricht der Operation ist ein XML Dokument, das dem CRL Schema entspricht. Das CRL Schema wird für diesen Zweck zusätzlich im types Bereich des WSDL Dokuments importiert und für die Deklaration der Rückgabe verwendet.

Transformation

Das ContextManipulationRequest Format wird in das AWQL Format umgewandelt. Dafür wird in einem Stylesheet ein neues AWQL Dokument angelegt, in das zusätzlich die nötigen Namensräume von Nexus eingefügt werden. Außerdem werden die Inhalte der Elemente *aa*, *ecs*, *nearestNeighbor*, *restriction* und *modifyingElement* in das neue AWQL Dokument übertragen.

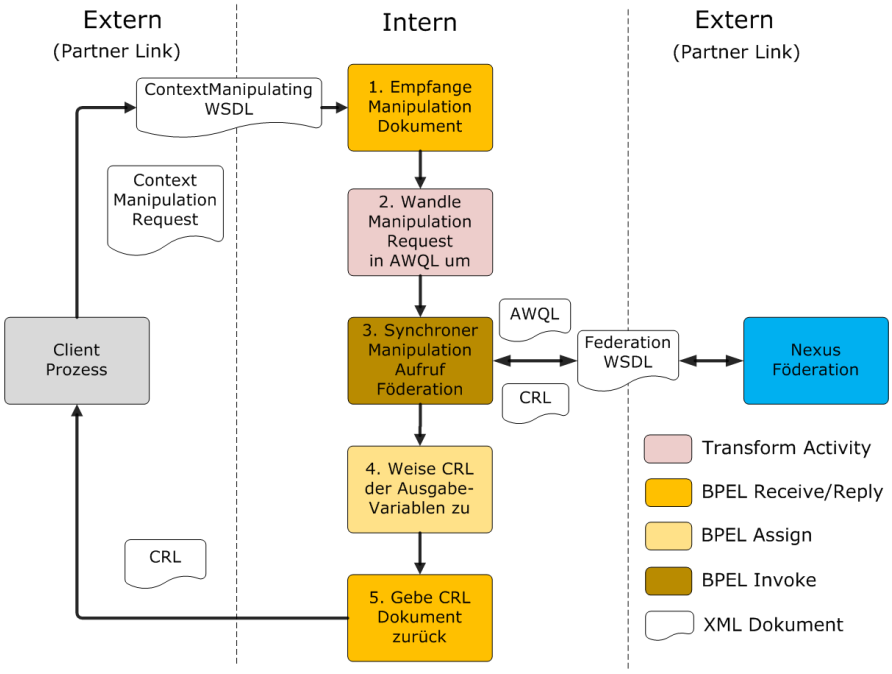

Abbildung 4.10: Prozessmodell von *ContextManipulation*

ContextManipulationAppend: Das Hinzufügen neuer Attribute zu bestehenden Objekten erfolgt mit dem Integrationsprozess *ContextManipulationAppend*, der den übergeordneten Prozess ContextManipulation vereinfacht. ContextManipulationAppend definiert das Nachrichtenformat ContextManipulationAppendRequest. Das Format erwartet Werte für die folgenden Elemente: *aa*, *ecs*, *nearestNeighbor* und *restriction* aus dem AWQL Schema und das *NexusAttribute* Element aus dem NSAS. In *NexusAttribute* wird das Attribut übergeben, das eingefügt werden soll. Das generische Element NexusAttribute wird verwendet, damit verschiedene Attributtypen eingetragen werden können (siehe Anhang A.1). Das Prozessmodell ist ähnlich wie bei der übergeordneten Schnittstelle ContextManipulation. Im Transformationsschritt wird hier jedoch ein Dokument des Typs ContextManipulationAppendRequest in ein Dokument für den Integrationsprozess *ContextManipulation* umgewandelt, also in das ContextManipulationRequest Format. Dafür wird ein neues Dokument vom Typ ContextManipulationAppendRequest erzeugt, in das alle Elemente aus dem ContextManipulationRequest übertragen werden. Dabei wird das NexusAttribute Element von einem append Element des AWQL Schemas eingeschlossen und dem modifyingElement eines ContextManipulationRequest Formats zugewiesen.
Ein anwendungsspezifischer Integrationsprozess auf der dritten Ebene der Integrationsprozesse (siehe Abb. 4.4) könnte ContextManipulationAppend so erweitern, dass anstatt dem generischen *NexusAttribute* ein spezielles Attribut definiert wird.
In Listing 4.8 sehen wir eine Beispielanfrage an den Integrationsprozess ContextManipulationAppend. Es soll das Attribut *description* dem Objekt mit dem Namen „Lagerhalle XY" hinzugefügt werden. Dafür wird eine entsprechende Auswahlbedingung mitgesendet, mit der das Objekt gefunden werden soll. Das Attribut description ist aus dem NSAS.
Nachdem das Attribut hinzugefügt worden ist, wird von ContextManipulationAppend eine entsprechende Antwort im CRL Format zurückgesendet. Die Antwort ist in Listing 4.9 dargestellt. Hierbei wird der *StaticObjectLocator* des veränderten Objekts entweder in der *success* Liste aufgeführt, falls das Einfügen des Attributs erfolgreich war, ansonsten in der *failure* Liste.

Listing 4.8: Beispielanfrage ContextManipulationAppend

```
<ContextManipulationAppendProcessRequest
 xmlns="http://nexus.informatik.uni-stuttgart.de/Federation/ContextManipulationAppend">
  <restriction><equal>
     <target>nsas:name.nsas:value</target>
     <referenceValue>Lagerhalle XY</referenceValue></equal>
  </restriction>
  <NexusAttribute>
   <nsas:description><nsas:value>Getriebeteilelager</nsas:value></nsas:description>
  </NexusAttribute>
</ContextManipulationAppendProcessRequest>
```

ContextManipulationUpdate: Das Aktualisieren von Objektattributen wird mit dem ContextManipulationUpdate Integrationsprozess vollzogen. Der Integrationsprozess ist identisch mit ContextManipulationAppend, bis auf die Bezeichnung für das Nachrichtenformat. Das Dokument vom Format ContextManipulationUpdateRequest definiert die gleichen Elemente wie das Format ContextManipulationAppendRequest. Die Transformation vom ContextManipulationUpdateRequest nach ContextManipulationRequest ist leicht abgeändert, da jetzt die Werte aus dem NexusAttribute Element in ein update Element aus dem AWQL Schema eingeschlossen werden. Als Resultat wird vom ContextManipulation ebenfalls ein CRL Dokument erwartet, dass an den Aufrufer weitergeleitet wird.
Für ein Beispiel kann die Bezeichnung des Dokumentformats aus Listing 4.8 in *ContextManipulationUpdateRequest* abgeändert werden. Daraufhin muss noch im description Element der Wert eingesetzt werden, der ab sofort den alten Wert ersetzen soll. Ein CRL Dokument liefert dann wieder das Ergebnis der Aktualisierung.

ContextManipulationDelete: Mit der Schnittstelle ContextManipulationDelete werden komplette Objekte aus der Augmented World gelöscht. Der Integrationsprozess legt ebenfalls ein eigenes Dokumentformat fest, das ContextManipulationDeleteRequest Format. In diesem Format wird lediglich das *restriction* Element aus dem AWQL Schema definiert, mit dem die zu löschenden Objekte ausgewählt werden. Das Prozessmodell ist identisch zu den bisherigen Schnittstellen ContextManipulationAppend und ContextManipulationUpdate. Im Transformationsschritt wird ein neues Dokument vom Format ContextManipulationRequest erzeugt. In dieses wird das restriction Element aus dem ContextManipulationDeleteRequest eingefügt und es wird zusätzlich ein delete Element aus dem AWQL Schema eingesetzt.
Ein Beispiel für eine Anfrage an die ContextManipulationDelete Schnittstelle kann ebenfalls aus Listing 4.8 abgeleitet werden. Dabei muss wieder lediglich der Bezeichner des Dokumentformats von ContextManipulationAppendRequest in ContextManipulationDeleteRequest geän-

Listing 4.9: Beispielantwort ContextManipulationAppend

```
<crl:changereport
 xmlns:crl="http://www.nexus.uni-stuttgart.de/2.0/CRL">
  <crl:success>
   <crl:object>(...StaticObjectLocator Lagerhalle XY...)</crl:object>
  </crl:success>
 <crl:failure></crl:failure>
</crl:changereport>
```

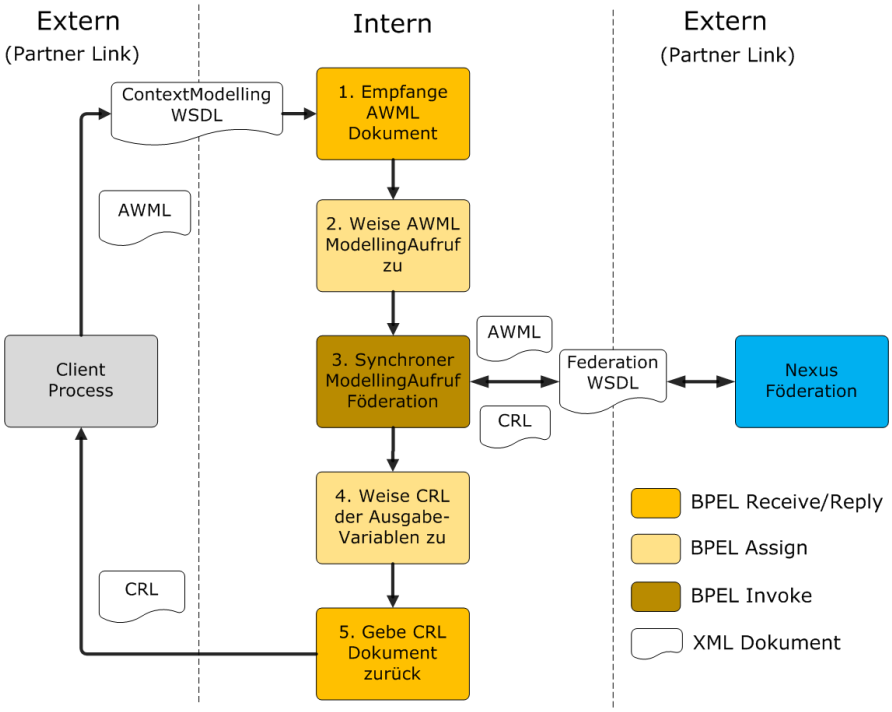

Abbildung 4.11: Prozessmodell von *ContextModeling*

dert werden. Außerdem kann bis auf das restriction Element alles aus der Anfrage entfernt werden. Nach dem Absenden der Anfrage an den ContextManipulationDelete Prozess werden Objekte mit dem Namen „Lagerhalle XY" aus der Augmented World Model entfernt. Das Resultat der Operation wird vom ContextManipulation Integrationsprozess in einem CRL Dokument mitgeteilt.

4.3.3 ContextModeling

Mit *ContextModeling* können neue Objekte der Augmented World hinzugefügt werden. Bei einer Anfrage über diesen Integrationsprozess werden Daten an die Föderation im AWML Format gesendet. Das Resultat der Anfrage wird im CRL Format erwartet und an den Aufrufer weitergeleitet.

Prozessmodellierung

In Abbildung 4.11 sehen wird das Prozessmodell für ContextModeling. Ein übergeordneter Prozess greift auf die Funktionalität von ContextModeling über dessen WSDL Beschreibung zu. Dabei erwartet ContextModeling als Eingabe ein Dokument im AWML Format. Der Aufruf wird an die Föderation weitergeleitet. Die Nachricht, die als Resultat von der Föderation übermittelt wird, ist im CRL Format (siehe Datentypen). Die Nachricht wird vom ContextModeling Prozess direkt an den Aufrufer zurückgegeben.

Listing 4.10: Beispielanfrage ContextModeling

```
<awml
 xmlns="http://www.nexus.uni-stuttgart.de/2.0/AWML"
 xmlns:nsas="http://www.nexus.uni-stuttgart.de/1.0/NSAS"
 xmlns:nscs="http://www.nexus.uni-stuttgart.de/1.0/NSCS">
  <nexusobject>
   <nsas:nol>
    <nsas:value>(...neuer StaticObjectLocator...)</nsas:value>
   </nsas:nol>
   <nsas:type>
    <nsas:value>nscs:Building</nsas:value>
   </nsas:type>
   <nsas:name>
    <nsas:value>Montagehalle ABC</nsas:value>
   </nsas:name>
   <nsas:address>
    <nsas:value>
     <nsas:street>Firmengasse</nsas:street>
     <nsas:number>123</nsas:number>
     <nsas:city>Muenchen</nsas:city>
     <nsas:zipCode>80805</nsas:zipCode>
    </nsas:value>
   </nsas:address>
   <nsas:description>
    <nsas:value>Fertigung von Motoren des Typs ABC</nsas:value>
   </nsas:description>
  </nexusobject>
</awml>
```

Web Service Schnittstelle

Für die Anfrage durch den Prozess ContextModeling wird in der WSDL-Beschreibung der Föderation die Operation *insertContextData* spezifiziert. Diese Operation erwartet zum Einfügen von Modelldaten in das Augmented World Model ein XML Dokument, das dem AWML Schema entspricht. Die Rückgabenachricht der Operation ist ein XML Dokument, das dem CRL Schema entspricht. Die beiden Schemata wurden bereits für die Integrationsprozesse ContextQuery und ContextManipulation im types Bereich der WSDL Beschreibung importiert.

Ausgetauschte Datentypen

Es werden sowohl zwischen dem aufrufenden Klienten-Prozess, als auch zwischen dem Context-Modeling Prozess und der Föderation, Nachrichten im Format AWML und CRL ausgetauscht. Für die Beschreibung der Datentypen, die dem AWQL und dem CRL Format entsprechen, werden bereits vorhandene XML Schemata verwendet (siehe Anhang A.4).

Transformation

Der Integrationsprozess *ContextModeling* sorgt für die Einbindung der komplexen Nachrichten-formate AWML und CRL, damit die vollständige Funktionalität für die Modellierung von Daten auf der Nexus Plattform bereitgestellt wird. Eine Transformation erfolgt in den Integrations-

prozessen auf der nächsten Ebene. Da die Modellierung von Daten jedoch anwendungsspezifisch ist, muss eine Verfeinerung dieses Integrationsprozesses für bestimmte Anwendungsbereiche gesondert erfolgen.

Beispiel für Anfrage- und Antwortnachricht

In Listing 4.10 ist eine Beispielanfrage im AWML Format aufgeführt, die der ContextModeling Prozess entgegennimmt. Bei der Anfrage muss der aufrufende Prozess sämtliche Namensräume der Schemata angeben, die er für die Modellierung der Daten verwendet. Hierfür sollten die entsprechenden Informationen aus Tabelle 4.1 genutzt werden. Die Beispielanfrage fügt der Augmented World ein neues Gebäude hinzu, das den Namen „Montagehalle ABC" trägt. Beim Einfügen neuer Objekte muss vom Modellierer ein StaticObjectLocator vergeben werden, der als eindeutiger Bezeichner für das Objekt dienen soll (siehe Anhang A.3). Zur Beschreibung des neuen Objekts werden Attribute aus dem NSAS und dem NSCS verwendet, weshalb die entsprechenden Namensräume mit aufgeführt sind. Die Antwort der Föderation auf das Einfügen dieses Objekts ist vergleichbar mit Listing 4.9, wobei nach erfolgreichem Einfügen der Daten der StaticObjectLocator des neuen Objekts in der success Liste zurückgegeben wird.

4.3.4 ContextEvent

Über die Schnittstelle ContextEvent können Ereignisse beim Ereignisdienst der Nexus Plattform registriert werden. Zur Registrierung eines Ereignisses werden Nachrichten im ERL Format erwartet und Benachrichtigungen über Ereignisse werden im ENL Format zurückgegeben.

Web Service Schnittstelle

Die Web Service Schnittstelle für den Ereignisdienst muss folgende Voraussetzungen erfüllen:

- Verarbeitung von Nachrichten im ERL, ENL und EventInfo Format
- Registrierung und Deregistrierung von Ereignissen
- Asynchrone Benachrichtigung über eingetretene Ereignisse
- Aufrufe über SOAP/HTTP

Für die Schnittstelle wird eine WSDL Beschreibung erstellt, die zwei Operationen enthält: registerEventOperation und deregisterEventOperation. Mit registerEventOperation lassen sich neue Ereignisse beim Ereignisdienst registrieren. Die Operation erwartet ein Dokument im ERL Format und liefert als Rückgabe ein Dokument im EventInfo Format. Für das ERL Format und das EventInfo Format existieren bereits Beschreibungen in XML Schema (siehe Anhang A.6), die für die Definition der Nachrichten für die registerEventOperation im types Bereich der WSDL importiert werden. Die registerEventOperation bündelt dabei gleich mehrere Funktionen des Ereignisdiensts:

- Registrierung des Ereignisses bei der Ereignisquelle (Event Source).
- Anmeldung für den Erhalt von Benachrichtigungen beim Benachrichtigungsdienst (Notification Service).
- Initialisierung der Beobachtung bei der Ereignisquelle.

Mit deregisterEventOperation lassen sich Ereignisse beim Ereignisdienst abmelden. Die Operation benötigt lediglich ein EventInfo Dokument als Eingabe und liefert keine Rückgabe. In der WSDL Beschreibung wird ein partnerLink Element für BPEL Prozesse deklariert, in dem der Ereignisdienst für die register und deregister Operationen die Rolle des Dienstanbieters erhält (EventServiceProvider).
Schließlich wird noch die Deklaration für die Ereignisbenachrichtigung benötigt. Die Ereignisbenachrichtigung beim Ereignisdienst ist asynchron. Für diesen Zweck werden in der WSDL Beschreibung drei Bestandteile eingefügt:

- Eine Callback Schnittstelle mit der Operation receiveNotification, die als Eingabe ein Dokument im ENL Format erwartet.
- Importieren des ENL Schemas im types Bereich zur Definition des Nachrichtenformats für die receiveNotification Operation.
- Hinzufügen einer Rolle für den Aufrufer der Operation receiveNotification in der Definition des partnerLinks (EventServiceRequester).

Die Operation *receiveNotification* wird dabei nicht von der Web Service Schnittstelle implementiert (siehe Abschnitt 4.4.2). Die Callback-Spezifikation dient nur dazu, dem Aufrufer mitzuteilen, dass Benachrichtigungen von der Web Service Schnittstelle asynchron versendet werden und welches Dokumentformat dabei verwendet wird. Die Implementierung der Methode, die in der Callback Schnittstelle angegeben wird, geschieht auf der Seite des aufrufenden Prozesses. Dort muss für den Empfang von Benachrichtigungen in einer Methode mit der Bezeichnung *receiveNotification* auf Nachrichten im ENL Format gewartet werden. Für diese asynchrone Verarbeitung wird zusätzlich dem Aufrufer im partnerLink Element die Rolle des *EventServiceRequester* zugewiesen. Die Rolle bindet den Aufrufer an die Implementierung der Callback Operation receiveNotification.
Die Zuordnung von Benachrichtigungen zu Workflowinstanzen erfolgt über die WS-Addressing Spezifikation. Dafür werden folgende Elemente aus der WS-Addressing Spezifikation in der WSDL-Beschreibung deklariert und im binding Bereich der Operation registerEventOperation zugewiesen:

- MessageID: Enthält einen Bezeichner, der vom aufrufenden Prozess vergeben wird und mit dem eine eindeutige Zuordnung der Nachricht auf Seiten des Aufrufers erfolgen kann.
- ReplyTo: Enthält die Endpunktadresse des aufrufenden Prozesses für den Rückruf durch die Web Service Schnittstelle.

Diese Callback Informationen des aufrufenden Prozesses werden bei der Registrierung eines Ereignisses in der Web Service Schnittstelle mit dem eindeutigen Bezeichner des Ereignisses (PredicateLocator, siehe Anhang A.3) in Beziehung gesetzt. Tritt ein Ereignis ein, so können die Callback Informationen über den eindeutigen Bezeichner des Ereignisses ermittelt werden. Die Web Service Schnittstelle versendet für die Benachrichtigung eine SOAP Nachricht, die im Body ein ENL Dokument trägt und an die Adresse aus dem ReplyTo Element der Callback Daten gerichtet ist. Die Zuordnung der Nachricht auf der Seite des aufrufenden Prozesses erfolgt über den Wert im MessageID Element. Dafür wird das RelatesTo Element aus der WS-Addressing Spezifikation in die WSDL Beschreibung hinzugefügt und der receiveNotification Operation im binding Bereich zugewiesen. Die Web Service Schnittstelle sorgt dafür, dass der Wert aus dem MessageID Element in das RelatesTo Element der Antwortnachricht übertragen wird. Für die Operationen registerEventOperation und receiveNotification wird im binding Bereich der WSDL zusätzlich der Versand über SOAP/HTTP im Nachrichtenstil Document/Literal festgelegt.

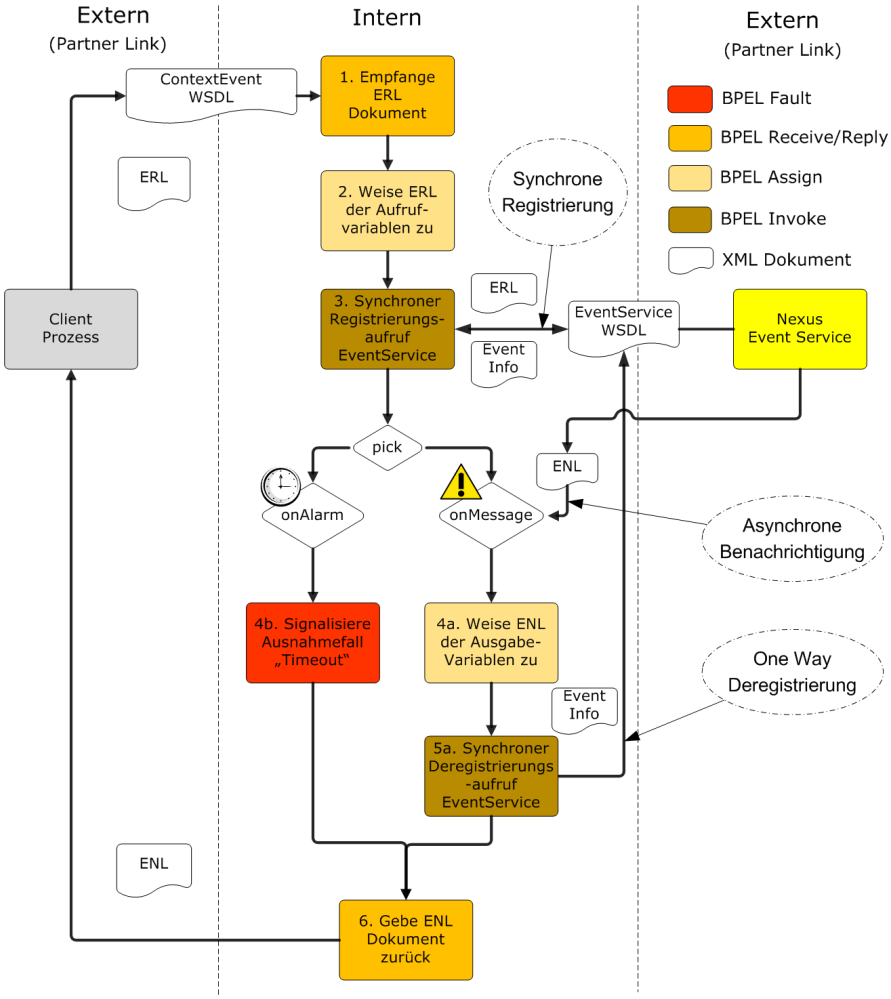

Abbildung 4.12: Prozessmodell von *ContextEvent*

Die Dokumentformate ERL, ENL und EventInfo werden im types Bereich der WSDL Beschreibung importiert (siehe Tabelle 4.1). In der WSDL Beschreibung gibt es zwei *service* Definitionen: eine für den Callback und eine für den Registrierungs- bzw. Deregistrierungsaufruf. Da die Callback Operation nicht von der Web Service Schnittstelle implementiert wird (s.o.), wird lediglich für die Schnittstelle mit den Operationen registerEventOperation und deregisterEventOperation eine Adresse für den Endpunkt der Implementierung angegeben.

Ausgetauschte Datentypen

Es werden sowohl zwischen dem aufrufenden Prozess, als auch zwischen dem ContextEvent Integrationsprozess und der Web Service Schnittstelle zum Ereignisdienst, Nachrichten im Format ERL, EventInfo und ENL ausgetauscht. Für die Beschreibung der Datentypen, die dem ERL, EventInfo und ENL Format entsprechen, werden bereits vorhandene XML Schemata verwendet (siehe Anhang A.6).

Listing 4.11: Beispielanfrage ContextEvent

```
<predicate
  xmlns="http://nexus.informatik.uni-stuttgart.de/EventRegistrationLanguage">
    <templateID>(...TemplateLocator für OnEnterArea...)</templateID>
    <name>ArbeiterBetrittArbeitsbereichXY</name>
    <thresholdProbability>0.9</thresholdProbability>
    <parameterList>
      <Parameter>
        <name>object identifier</name>
        <type>de.uni_stuttgart.nexus.types.basic.ObjectId</type>
        <value>(...UUID des Arbeiters...)</value>
      </Parameter>
      <Parameter>
        <name>observation area</name>
        <type>de.uni_stuttgart.nexus.types.location.Area</type>
        <value>(..Gebietsangabe des Arbeitsbereichs im WKT Format..)</value>
      </Parameter>
    </parameterList>
    <predicateManagementParameterList>
      <predicateManagementParameter>
        <name>registration interval</name>
        <type>java.lang.Long</type>
        <value>120</value>
      </predicateManagementParameter>
    </predicateManagementParameterList>
</predicate>
```

Prozessmodellierung

In Abbildung 4.12 sehen wird das Prozessmodell für ContextEvent. Der Prozess stellt seinen Dienst über eine WSDL Beschreibung nach außen hin zur Verfügung. Der übergeordneten Workflowanwendung wird der ContextEvent Integrationsprozess als asynchroner Dienst angeboten, da die Web Service Schnittstelle zum Ereignisdienst ebenfalls über einen asynchronen Teil verfügt (siehe Web Service Schnittstelle). Dadurch wird die übergeordnete Workflowanwendung nach einem Aufruf von ContextEvent nicht blockiert und kann parallel andere Aktionen durchführen. ContextEvent leitet die Eingabe im ERL Format an den Aufruf des Ereignisdiensts weiter. Die Kommunikation zum Ereignisdienst teilt sich beim regulären Ablauf in drei Schritte auf:

- Synchroner Aufruf zur Registrierung eines Ereignisses (Schritt 3 im Prozessmodell).
- Empfang der asynchronen Benachrichtigung vom Ereignisdienst im onMessage Element (onMessage Zweig des Prozesses).
- Deregistrierungsaufruf nach Erhalt der Benachrichtigung (Schritt 5a.).

Für die Registrierung wird das ERL Dokument an den Ereignisdienst weitergeleitet. Die erfolgreiche Registrierung des Ereignisses wird vom Ereignisdienst durch eine EventInfo Nachricht quittiert. Die asynchrone Benachrichtigung über ein eingetretenes Ereignis erfolgt im onMessage Zweig des Prozesses mit einer ENL Nachricht. Das onMessage Element implementiert folglich die Callback Funktion receiveNotification (siehe Web Service Schnittstelle). Danach erfolgt die Deregistrierung des Ereignisses (Schritt 5a.) mit dem EventInfo Dokument aus Schritt 3 und die Weiterleitung der Benachrichtigung im ENL Format an den aufrufenden Prozess.
Es gibt auch einen alternativen Prozessablauf, da bei einer Registrierung immer auch die Gültig-

Listing 4.12: Beispielantwort ContextEvent

```
<notification
  xmlns="http://nexus.informatik.uni-stuttgart.de/EventNotificationLanguage">
    <id>(...NotificationLocator...)</id>
    <predicateId>(...PredicateLocator...)</predicateId>
    <templateId>(...TemplateLocator...)</templateId>
    <name>OnEnterAreaObsModule</name>
    <service>Location Service</service>
    <server>(...Adressdaten Server...)</server>
    <counter>1</counter>
    <scope>(...Zeitstempel Gültigkeitsdauer...)</scope>
    <timestamp>(...Beobachtungszeit...)</timestamp>
    <comment>Basic Event</comment>
    <variableList>
      <variable>
        <name>Entering Object</name>
        <type>de.uni_stuttgart.nexus.types.basic.OInfo</type>
        <restriction>Mobile Object</restriction>
        <value>(... Daten zum Objekt...)</value>
      </variable>
      <variable>
        <name>Entered Space</name>
        <type>de.uni_stuttgart.nexus.types.location.Segment</type>
        <restriction/>
        <value>(...Daten zum Gebiet...)</value>
      </variable>
    </variableList>
</notification>
```

keitsdauer des Ereignisses mit angegeben wird (siehe Anhang A.6). Deshalb gibt es den onAlarm Zweig im Prozess, der nach Ablauf der Gültigkeitsdauer aufgerufen wird. Eine Deregistrierung des Ereignisses ist in diesem Zweig nicht erforderlich, da dies automatisch vom Ereignisdienst erledigt wird. Der Prozess erzeugt in diesem Fall eine Ausnahme vom Typ *ContextEventFault-Message*. Die übergeordnete Workflowanwendung sollte diese Ausnahme entsprechend auffangen und behandeln.

Transformation

Der Integrationsprozess *ContextEvent* sorgt für die Einbindung der komplexen Nachrichtenformate ERL, ENL und EventInfo, um die vollständige Funktionalität zur Registrierung von Ereignissen auf der Nexus Plattform bereitzustellen. Beim ContextEvent Prozess wird keine Transformation von Nachrichten durchgeführt. Eine Transformation wird z.B. in den Integrationsprozessen auf der zweiten Ebene durchgeführt (siehe ContextEventOnEnterArea 4.3.4).

Beispiel für Anfrage- und Antwortnachricht

In Listing 4.11 sehen wir eine Beispielanfrage an die ContextEvent Schnittstelle im ERL Format. Mit der Anfrage soll beobachtet werden, ob ein Arbeiter einen bestimmten Arbeitsbereich betritt. Dafür wurde dem Ereignis im *name* Element ein verständlicher Name gegeben. Mit dem Wert in threshold Probability wird festgelegt, dass sich der Ereignisdienst zu 90 % Wahrscheinlichkeit

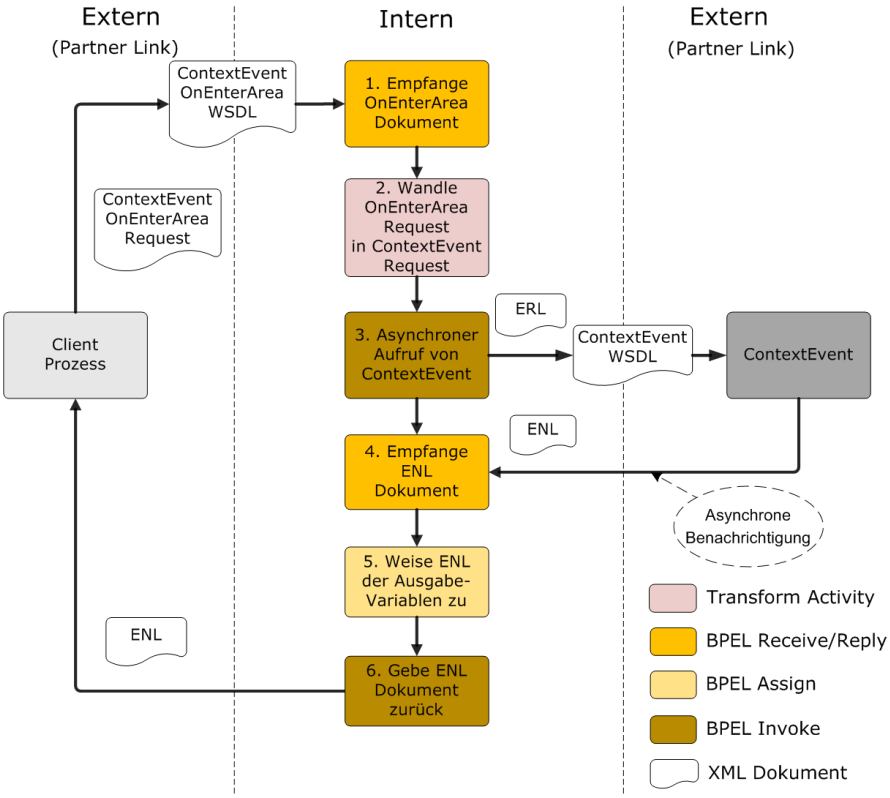

Abbildung 4.13: Prozessmodell von *ContextEventOnEnterArea*

sicher sein sollte, dass das Ereignis tatsächlich eingetreten ist, bevor eine Benachrichtigung versendet wird. In der Parameterliste wird nun noch der Arbeiter durch seine eindeutige ID festgelegt und der Arbeitsbereich durch die Angabe der geografischen Daten. Für die Gültigkeit des Ereignisses wurden 120 Sekunden angegeben. Details zu den einzelnen Elementen befinden sich in Abschnitt A.6. In Listing 4.12 sehen wir eine entsprechende Antwort des Ereignisdiensts nach Eintritt des Ereignisses. Im Element variableList sind dabei die Variablen eingetragen, die beobachtet wurden.

ContextEventOnEnterArea: Für die Registrierung verschiedener Ereignistypen können zusätzlich verfeinerte Integrationsprozesse erstellt werden. ContextEventOnEnterArea wurde beispielhaft als Integrationsprozess der zweiten Ebene für Ereignisse vom Typ On Enter Area erstellt. Das Prozessmodell wird in Abbildung 4.13 dargestellt. Die ContextEventOnEnterArea Schnittstelle definiert das einfachere Format ContextEventOnEnterAreaRequest mit folgenden Elementen:

- **name**: Dieses Element ist vom XML Schema Typ *string* und nimmt einen beliebigen Namen für das Ereignis an.
- **enteringObject**: In diesem Element wird der Bezeichner des Objekts (siehe Anhang A.3) übergeben, dessen Eintritt in ein Gebiet beobachtet werden soll. Das Element ist ebenfalls vom XML Schema Typ *string*.
- **area**: Hier wird die Gebietsangabe im WKT Format erwartet. Das Element ist vom XML Schema Typ *string*.

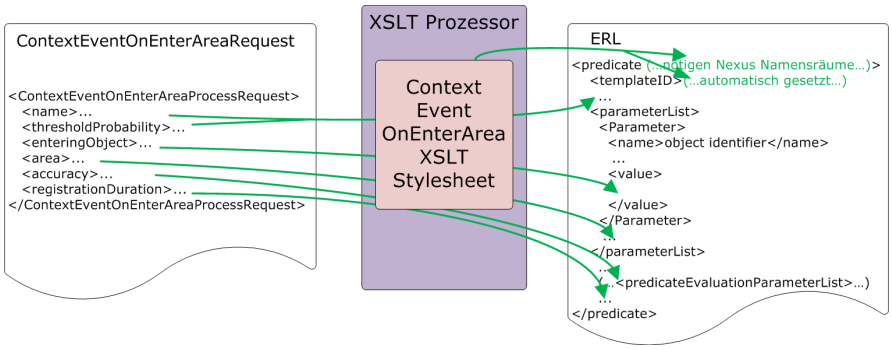

Abbildung 4.14: Transformation von ContextEventOnEnterAreaRequest nach ERL

- **thresholdProbability**: Mit diesem Element wird die Schwelle für den Wahrscheinlichkeitswert angegeben. Das Element ist vom XML Schema Typ *float*.
- **accuracy**: Mit accuracy kann die Genauigkeit der Positionsbestimmung in Metern angegeben werden. Das Element ist vom XML Schema Typ *double*.
- **registrationDuration**: Mit dem letzten Element wird die Gültigkeitsdauer der Registrierung in Sekunden angegeben. Das Element ist vom XML Schema Typ *long*.

In Schritt 2 wird eine Transformation des ContextEventOnEnterAreaRequest Dokuments in ein ERL Dokument für den ContextEvent Integrationsprozess vorgenommen (siehe Abb. 4.14). Dabei werden z.B. die Daten aus dem Element accuracy in das Element predicateEvaluationList des ERL Dokuments aufgenommen (siehe Anhang A.6). Die fehlenden Daten, wie z.B. der Bezeichner für *On Enter Area* Ereignisse (templateID), werden vom XSLT Stylesheet für die ContextEventOnEnterArea Schnittstelle automatisch erzeugt. Die asynchrone Benachrichtigung des Ereignisdiensts erfolgt in Schritt 4.

In Listing 4.13 ist eine Beispielanfrage an den ContextEventOnEnterArea Integrationsprozess formuliert. Die Rückgabe von ContextEvent entspricht dabei Listing 4.12. Es wird deutlich, dass ContextEventOnEnterArea ein einfacheres Format verwendet, als ContextEvent (siehe Listing 4.11). Dafür können lediglich On Enter Area Ereignisse registriert werden.

4.3.5 GetMap

Für die Erstellung von geografischen Karten wird die Schnittstelle *GetMap* entworfen. Bei einer Anfrage über diese Schnittstelle werden Daten an den Kartendienst der Nexus Plattform im MapPL Format gesendet. Das Resultat der Anfrage wird im MapML Format erwartet.

Listing 4.13: Beispielanfrage ContextEventOnEnterArea

```
<ContextEventOnEnterAreaProcessRequest
 xmlns="http://nexus.informatik.uni-stuttgart.de/EventService/ContextEventOnEnterArea">
  <name>ArbeiterBetrittArbeitsbereichXY</name>
  <enteringObject>(...UUID des Arbeiters...)</enteringObject>
  <area>(...Gebietsangabe des Arbeitsbereichs im WKT Format als String...)</area>
  <thresholdProbability>0.9</thresholdProbability>
  <accuracy>10</accuracy>
  <registrationDuration>120</registrationDuration>
</ContextEventOnEnterAreaProcessRequest>
```

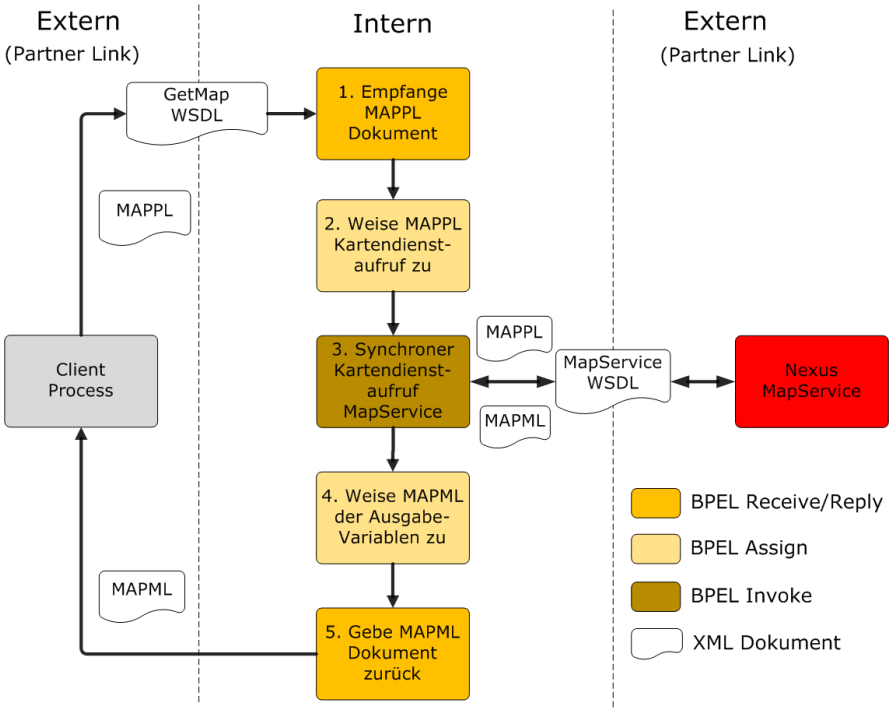

Abbildung 4.15: Prozessmodell von *GetMap*

Prozessmodellierung

In Abbildung 4.15 sehen wird das Prozessmodell für die Schnittstelle GetMap. Der Prozess stellt seinen Dienst über eine eigene WSDL nach außen hin zur Verfügung. Dabei erwartet GetMap als Eingabe ein Dokument im MapPL Format. Die Eingabe wird dann dem Aufruf an den Kartendienst als Parameter zugewiesen. Die Dienste des Kartendiensts werden ebenfalls über eine WSDL-Beschreibung in Anspruch genommen (s.u.). Der Aufruf des Kartendiensts läuft synchron ab. Die Nachricht, die als Resultat vom Kartendienst übermittelt wird, ist vom MapML Format. Die Nachricht wird vom GetMap Prozess direkt an den Aufrufer zurückgegeben.

Ausgetauschte Datentypen

Es werden sowohl zwischen dem aufrufenden Klienten-Prozess, als auch zwischen der GetMap Schnittstelle und dem Kartendienst, Nachrichten im Format MapPL und MapML ausgetauscht. Die Schnittstellen sind also identisch. Für die Beschreibung der Datentypen, die dem MapPL und dem MapML Format entsprechen, werden bereits vorhandene XML Schemata verwendet (siehe Anhang A.7).

Web Service Schnittstelle

Für die Verarbeitung auf der Nexus Plattform existiert bereits ein Servlet, welches Anfragen im MapPL Format verarbeiten kann. Als Resultat sendet das Servlet Kartendaten im MapML Format. Dabei handelt es sich um das gleiche Servlet, das bei den Schnittstellen ContextQuery, ContextManipulation und ContextModeling Anfragen für die Föderation entgegennimmt. Für

Listing 4.14: Beispielanfrage GetMap

```
<mappl xmlns="http://www.nexus.uni-stuttgart.de/2.0/MapPL" ...>
 <imageSize width="800" height="600"/>
 <imageEncoding format="SVG"/>
 <clippingRectangle>
  <bottomLeftCorner>(...Punkt im WKT Format...)</bottomLeftCorner>
  <topRightCorner>(...Punkt im WKT Format...)</topRightCorner>
 </clippingRectangle>
 <drawingRule>
  <nexusType>nscs:Building</nexusType>
  <geoElement styleClass="BuildingExtent">
   <awql:target>nsas:extent.nsas:value</awql:target>
   <textElement styleClass="BuildingName">
    <awql:target>nsas:name.nsas:value</awql:target>
   </textElement>
  </geoElement>
 </drawingRule>
 <css>
   path.CPolygon-BuildingExtent {fill: orange}
   text.BuildingName {font-size: 24}
 </css>
 <awql xmlns="http://www.nexus.uni-stuttgart.de/2.0/AWQL" ...>
  <restriction>
    <intersects><target>nsas:extent.nsas:value</target>
     <referenceValue>(...Gebiet im WKT Format...)</referenceValue>
    </intersects>
  </restriction>
 </awql>
</mappl>
```

diesen Dienst wird jedoch eine separate WSDL-Beschreibung erzeugt, die für den Kartendienst den Nachrichtenstil Document/Literal spezifiziert. Für die Anfrage durch den Prozess GetMap wird in der WSDL-Beschreibung die Operation *getMapData* spezifiziert. Diese Operation erwartet zum Generieren von Karten ein XML Dokument, das dem MapPL Schema entspricht. Die Rückgabenachricht der Operation ist ein XML Dokument das dem MapML Schema entspricht. Die vorhandenen Schema Dateien werden deshalb in den types Bereich des WSDL Dokuments importiert und für die Deklaration der Nachrichten verwendet. Die Dokumente im MapPL- und MapML-Format werden über SOAP/HTTP versendet, deshalb wird für die getMapData Operation ein SOAP-Binding deklariert. Im service Element der WSDL-Beschreibung wird zusätzlich die Adresse des Servers eingetragen, an der das Servlet die Anfragen entgegennimmt. Aufrufe externer Dienste durch BPEL Prozesse werden über eine Partnerverbindung (partnerLink) realisiert. In der WSDL-Beschreibung wird ein partnerLink erzeugt, in dem der Kartendienst die Rolle des Dienstanbieters erhält (MapServiceProvider).

Transformation

Der Integrationsprozess *GetMap* sorgt für die Einbindung der komplexen Nachrichtenformate MapPL und MapML, um die vollständige Funktionalität für die Generierung von Karten zur Verfügung zu stellen. Er nimmt folglich keine Transformation von Ein- und Ausgabeformaten vor. Es wäre denkbar, für den Kartendienst einen verfeinerten Integrationsprozess auf der zweiten

Listing 4.15: SVG Dokument aus dem Body eines MapML Dokuments

```
<svg xmlns="http://www.w3.org/2000/svg" ... >
  <style type="text/css" ... >
    path.CPolygon-BuildingExtent {fill: orange}
    text.BuildingName {font-size: 24}
  </style>
  <g>
    <g id="(...StaticObjectLocator/GebäudeXY...)">
      <path d="(...SVG-Linienzug...)" id="id2"
                fill-rule="evenodd" class="CPolygon-BuildingExtent"/>
      <text x="469.309" y="365.215">
    <tspan class="BuildingName">Fertigungshalle XY</tspan>
      </text>
    </g>
    <g id="(...StaticObjectLocator/GebäudeYZ...)">
      <path d="(...SVG-Linienzug...)" id="id4"
                fill-rule="evenodd" class="CPolygon-BuildingExtent"/>
      <text x="469.309" y="365.215">
        <tspan class="BuildingName">Lagerhalle YZ</tspan>
      </text>
    </g>
      ...
  </g>
</svg>
```

Ebene zu erstellen, bei dem gewisse Darstellungsparameter fest kodiert werden, z.B. Auflösung, Bildformat oder Darstellung der Objekte.

Beispiel für Anfrage- und Antwortnachricht

In Listing 4.14 ist eine Beispielanfrage im MapPL Format aufgeführt, die der GetMap Prozess entgegennimmt. Die Wahl der Namensräume von MapPL und MapML erfolgte anhand Tabelle 4.1. Die Beispielanfrage generiert die Karte eines spezifizierten Gebiets im SVG Format. Es ist eine Zeichenregel vorhanden, mit der Objekte der Klasse Building formatiert werden (siehe Anhang A.7). Im impliziten AWQL Dokument wird das Gebiet für die Erstellung der Karte ausgewählt.
Die Rückgabe enthält die entsprechenden Kartendaten im MapML Format, wobei im Body der MapML Nachricht ein Dokument im SVG Format enthalten ist (siehe Listing 4.15).

4.4 Implementierung

Im Grobentwurf wurde eine Lösung für die Einbindung von Nexusdiensten auf hoher Ebene erarbeitet, die dann im Feinentwurf durch die Ausarbeitung der Prozessmodelle, der Transformationen und der Web Service Schnittstellen verfeinert wurde. Dennoch wurden die Komponenten aus dem Feinentwurf, soweit es möglich war, unabhängig von den verwendeten Technologien beschrieben. In diesem Abschnitt soll deshalb auf die Implementierungsdetails bezüglich der verwendeten Technologien BPEL, WSDL, XML und XSLT eingegangen werden. Als Programmiersprache wird Java verwendet, da die Nexus Plattform ebenfalls mit Java implementiert wurde.

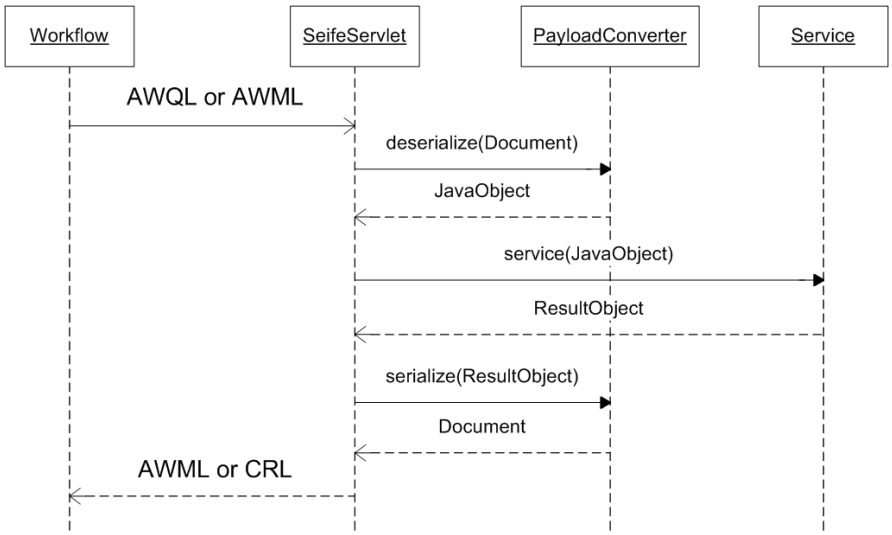

Abbildung 4.16: Sequenzdiagramm *Nachrichtenverarbeitung im SeifeServlet* (vereinfacht)

4.4.1 ContextModeling, ContextQuery und ContextManipulation

Die Integrationsprozesse ContextModeling, ContextQuery, ContextManipulation greifen über die Web Service Schnittstelle der Föderation auf die Nexus Plattform zu. Deshalb sollen sie hier auch gemeinsam betrachtet werden. Zuerst wird die Implementierung der Web Service Schnittstelle zur Nexus Plattform vorgestellt. Darauf folgen Implementierungsdetails zu den jeweiligen BPEL Prozessen der o.g. Integrationsprozesse. Es werden schließlich noch die Transformationsbeschreibungen und die XML Schemata für die Datentypen erstellt.

Web Service Implementierung

Für den Zugriff auf die Föderation wird für die Integrationsprozesse eine gemeinsame WSDL Beschreibung erzeugt, die in der Datei Federation.wsdl abgelegt wird (siehe Listing B.2). Der Endpunkt der Schnittstelle, der die Operationen implementiert und im service Bereich der WSDL deklariert ist, ist das SeifeServlet aus dem Projekt nexus–library–seife. In Abbildung 4.16 ist die Verarbeitung der Nachrichten im SeifeServlet vereinfacht dargestellt. Für AWQL und AWML Dokumente hält das SeifeServlet entsprechende Serialisierer und Deserialisierer bereit, mit denen die Dokumente in Java Objekte umgewandelt werden und umgekehrt (PayloadConverter). Nach der Deserialisierung wird das resultierende Java Objekt an einen Dienst gesendet (Service). Für AWQL und AWML Dokumente wird die Service-Klasse des SeifeServlets vom Föderationsdienst implementiert. Nach der Bearbeitung durch den Föderationsdienst wird das zurückgegebene Java Objekt wieder in ein AWML bzw. CRL Dokument serialisiert.

BPEL Prozesse

Für die Integrationsprozesse ContextQuery, ContextManipulation und ContextModeling, sowie für deren feinere Varianten, wie z.B. ContextQueryExclude, werden jeweils eigene BPEL Prozesse erzeugt (siehe Abschnitt 2.4.2). Die Schnittstelle dieser BPEL Prozesse ist über ein WSDL Dokument beschrieben (siehe Abschnitt 2.3.2). Die oben genannten Integrationsprozesse

werden alle als synchrone Prozesse realisiert. Für die Modellierung der Prozesse werden die entsprechenden Aktivitäten der im Feinentwurf vorgestellten Prozessmodelle durch Elemente aus der BPEL Spezifikation ersetzt [BPEL]. Für den Integrationsprozess ContextQuery wird dazu exemplarisch in Listing C.1 die BPEL Beschreibung und in Listing B.4 die WSDL Beschreibung aufgeführt. Für die ausgetauschten Dokumente werden Variablen im variables Element des Prozesses angelegt. Die Anfrage eines Klienten-Prozesses wird in einem receive Element entgegengenommen. Der Aufruf der Föderation wird durch ein invoke Element realisiert. Das invoke Element benötigt für die Ausführung der Aktivität neben der Ein- und Ausgabevariable, die Angabe des partnerLinks, der Schnittstelle und der Operation, die beim Partner aufgerufen wird. Dafür muss für den Web Service der Föderation ein partnerLink Element definiert werden, in dem die Föderation die Rolle des FederationServiceProvider erhält, also die Rolle des Anbieters von Föderationsdiensten. Die Verbindung zur WSDL Beschreibung der Föderation und dem deklarierten partnerLink im Prozess geschieht bei der Oracle BPEL Engine über einen separaten Deskriptor (bpel.xml, mehr dazu bei [Oracle]). Die Ausgabevariable im invoke Element speichert das Ergebnisdokument, das von der Föderation zurückgesendet wird. Das Ergebnis wird der Ausgabevariablen mit einem assign Element zugewiesen. Die Rückgabe an den Klienten-Prozess erfolgt über ein reply Element, in dem als partnerLink die Schnittstellenbeschreibung des aktuell aufgerufenen Prozesses angegeben wird. Sämtliche Aktivitäten (receive, assign, invoke, reply) werden von einem sequence Element umgeben, d.h. sie werden hintereinander ausgeführt.

XML Datentypen

Die Beschreibung der ausgetauschten Dokumente erfolgt mit XML Schema. Für die Dokumente AWQL, AWML und CRL werden dafür bereits existierende Schemata verwendet. Die Schemata sind in den jeweiligen Dateien NexusAwqlSchema.xsd, NexusAwmlSchema.xsd und NexusCrlSchema.xsd definiert und werden bei [NexusSchema] zur Verfügung gestellt.

XSLT Stylesheets

Für die Integrationsprozesse ContextQueryInclude, ContextQueryExclude, ContextManipulationAppend, ContextManipulationUpdate und ContextManipulationDelete werden XSL Stylesheets erzeugt, die für die Transformation der Dokumentformate verwendet werden (siehe Abschnitt 2.2.2). In Listing D.1 ist die Transformationsbeschreibung von ContextQueryExclude aufgeführt. Sie soll hier stellvertretend für die XSL Stylesheets der übrigen Prozesse exemplarisch vorgestellt werden. Die Transformationsbeschreibung wandelt Dokumente vom Format ContextQueryExcludeRequest (Quelldokument) in das Format des Integrationsprozesses ContextQuery. Das Template

```
<xsl:template match="/">
```

greift dabei auf die Wurzel des Quelldokuments zu. In ihm werden deshalb auch die Beginn- und Ende-Bezeichner des Zieldokuments deklariert (hier AWQL). Außerdem werden in diesem Template auch die Attribute im target Element des Quelldokuments verarbeitet. Für jedes Attribut aus dem Quelldokument wird im Zieldokument ein exclude Element aus dem AWQL Schema erstellt. Mit apply-templates werden die übrigen templates, die noch im Stylesheet definiert sind, auf den Rest des Quelldokuments angewendet. Zum Beispiel wird mit dem Template

```
<xsl:template match="awql:restriction">
```

die Behandlung für das restriction Element des Quelldokuments deklariert. Dabei wird das restriction Element aus dem Quelldokument mit dem Befehl

```
<xsl:copy-of select="*"/>
```

in das zu erzeugende Zieldokument kopiert, wobei die AWQL spezifischen Bezeichner um den kopierten Ausdruck herum gesetzt werden. Der Zugriff auf Teile des Quelldokuments erfolgt mit XPath Ausdrücken (siehe Abschnitt 2.2.3). Das Template

```
<xsl:template match="*">
```

wird auf alle Elemente angewendet, für die kein eigenes Template definiert wurde. Es sorgt dafür, dass bei diesen Elementen keine Aktion durchgeführt wird.
Die Definition der Stylesheets der übrigen Integrationsprozesse ist ähnlich, wobei jeweils die spezifischen Elemente der jeweiligen Quelldokumente behandelt werden müssen (siehe Abschnitt 4.3).
Rückgaben werden von den Integrationsprozessen direkt an den aufrufenden Prozess weitergegeben (z.B. im AWML Format). Der Modellierer muss deshalb im aufrufenden Prozess eine Transformation erzeugen, mit der er sämtliche Objekte und Attribute herausfiltert, die er für die weitere Verarbeitung benötigt. In Abschnitt 4.6 ist eine beispielhafte Workflowanwendung aufgeführt, bei der die Verarbeitung von AWML Dokumenten aufgezeigt wird.
Die Oracle BPEL Engine stellt für die Verarbeitung von XSLT Stylesheets einen XSLT Prozessor bereit, der über die Funktion *processXSLT* aufgerufen werden kann. Die Funktion erwartet als Eingabewerte das entsprechende Stylesheet und das Quelldokument. In Listing C.2 geschieht dies im BPEL Prozess von ContextQueryExclude im ersten assign Element.

Pakete

Der BPEL Prozess eines Integrationsprozesses wird zusammen mit seiner WSDL-Beschreibung und seinem Deskriptor (bpel.xml) zu einem Archiv hinzugefügt. Das Archiv enthält zusätzlich das Stylesheet, sofern eine Transformation im Prozess definiert wird. Das komplette Archiv wird dann auf der BPEL Engine installiert.
Die Implementierung der Web Service Schnittstelle zur Föderation ist bereits in der Föderationskomponente integriert (SeifeServlet). Für den Zugriff auf den Web Service wird die WSDL-Beschreibung des Föderationsdiensts (Federation.wsdl) an einem zugreifbaren Ort zur Verfügung gestellt. In der WSDL Beschreibung wird vorher noch der Endpunkt eingetragen, an dem das SeifeServlet die Anfragen für die Föderation entgegennimmt.

4.4.2 ContextEvent

Der Integrationsprozess ContextEvent und die verfeinerte Variante ContextEventOnEnterArea, greifen ebenfalls über eine gemeinsame Web Service Schnittstelle auf den Ereignisdienst der Nexus Plattform zu. Die Implementierung der Web Service Schnittstelle zur Nexus Plattform soll deshalb zuerst beschrieben werden. Darauf folgen Details zu den BPEL Prozessen, den XML Schemata für die Datentypen, den XSL Stylesheets und der Installation der Komponenten.

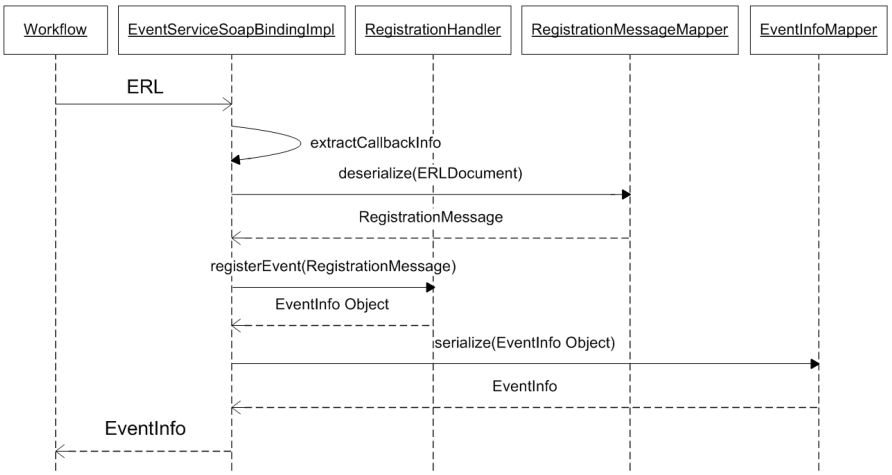

Abbildung 4.17: Sequenzdiagramm *Ereignisregistrierung über WS-Schnittstelle*

Web Service Implementierung

Für die Realisierung der Web Service Schnittstelle zum Ereignisdienst wird die SOAP Engine von Apache Axis 1.4 verwendet (siehe Abschnitt 2.3.2). Die Java Version von Axis implementiert für die Realisierung von Web Services die Standardschnittstellen *Java API for XML-based RPC* (JAX-RPC) und *SOAP with Attachments API for Java* (SAAJ) (siehe dazu auch [Eng93] und [CJ02]) und bietet deren Funktionalität in erweiterter Form dem Anwender über vereinfachte Schnittstellen zur Verfügung, z.B. zum einfachen Aufbau und Versand von SOAP Nachrichten. In Abbildung 4.17 wird in einem UML-Sequenzdiagramm [UML] die Verarbeitung einer Registrierungsanfrage gezeigt. Aufrufe der Workflowanwendung werden von der Axis Engine in einem Servlet entgegengenommen, welches dann die Anfrage an die Klasse EventServiceSoapBindingImpl weiterleitet. Sie nimmt Aufrufe im ERL Format für Ereignisregistrierungen und im EventInfo Format für Ereignisderegistrierungen entgegen. Bei einer Registrierung werden die Rückrufdaten aus dem WS-Addressing Header, die im binding Bereich der WSDL für SOAP Nachrichten definiert wurden (siehe Abschnitt 4.3.4), extrahiert. Zu diesem Zweck werden WS-Addressing Handler der Axis Umgebung verwendet (siehe Abschnitt 2.3.2). Für die Serialisierung und Deserialisierung von XML Daten werden die beiden Hilfsklassen RegistrationMessageMapper und EventInfoMapper implementiert. Die entsprechenden Java Objekte für ERL und EventInfo Dokumente sind bereits gegeben und lauten RegistrationMessage und EventInfo. Das RegistrationMessage Objekt wird zusammen mit den Rückrufdaten des Workflows an die Klasse RegistrationHandler übergeben. Der RegistrationHandler kümmert sich um den Aufbau der Kommunikation zum Ereignisdienst und das Registrierungsprozedere. In Abbildung 4.18 wird dazu ein weiteres Sequenzdiagramm präsentiert, in dem die Aktionen des RegistrationHandler dargestellt sind. Der RegistrationHandler führt dabei vier Schritte hintereinander durch:

- **Registrierung** des Ereignisses bei der Ereignisquelle. Hierfür wird die vorhandene Schnittstellenklasse EventSourceIf verwendet. Mit der Klasse NexusStubFactory kann für EventSourceIf ein Stub für die Kommunikation zur Ereignisquelle erzeugt werden.
- **Anmeldung** beim Benachrichtigungsdienst für den Empfang von Benachrichtigungen zum registrierten Ereignis. Dafür wird die vorhandene Schnittstellenklasse EC2NSIf (Event Client to Notification Service Interface) verwendet. Für diese Schnittstelle kann ebenfalls über die NexusStubFactory ein Stub für die Kommunikation zum Notification Service

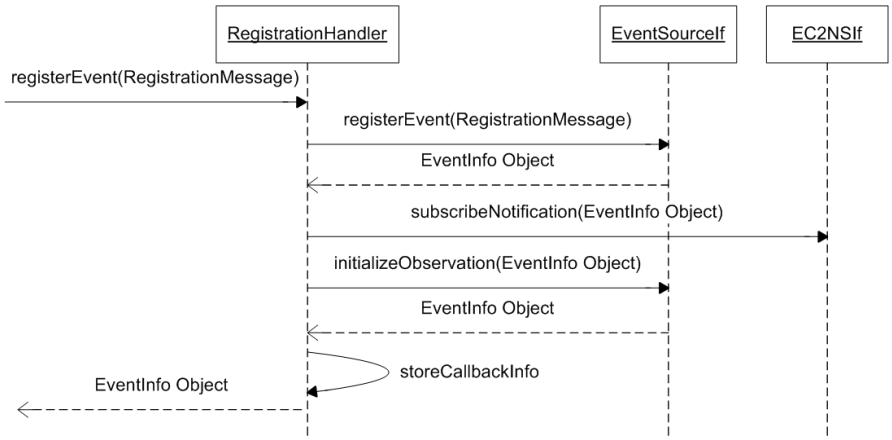

Abbildung 4.18: Sequenzdiagramm *Ereignisregistrierung im RegistrationHandler*

erzeugt werden.

- **Initialisierung** der Beobachtung des Ereignisses. Dies geschieht wieder über die Schnittstelle zur Ereignisquelle.
- **Speicherung** der Rückrufdaten des Workflows für den Fall, dass das Ereignis eintritt und der Workflow benachrichtigt werden soll.

Der Ereignisdienst erzeugt bei der Registrierung des Ereignisse ein EventInfo Objekt, das vor der Rückgabe an den Workflow (siehe Abb. 4.17) durch die Klasse EventInfoMapper in ein EventInfo Dokument serialisiert wird. Damit ist der Registrierungsaufruf zu Ende, wobei auf der Nexus Plattform das Ereignis weiterhin beobachtet wird.
Für Benachrichtigungen durch den Ereignisdienst muss der RegistrationHandler die vorhandene Schnittstelle EventClientIf implementieren. In Abbildung 4.19 sehen wir den Ablauf. Der RegistrationHandler nimmt die Benachrichtigung entgegen und holt die Rückrufdaten des Workflows aus dem Speicher. Danach gibt er die Informationen an die Klasse NotificationHandler weiter. Der NotificationHandler kümmert sich um den Versand der Benachrichtigungen an den entsprechenden Workflow über SOAP/HTTP im Document/Literal Nachrichtenstil. Er erstellt dazu aus dem NotificationMessage Objekt, das vom Benachrichtigungsdienst übergeben wurde, ein Dokument im ENL Format. Dies geschieht über die Hilfsklasse NotificationMapper. Anschließend wird das ENL Dokument über ein Call-Objekt, das von der Apache Axis Umgebung für SOAP-Aufrufe zur Verfügung gestellt wird, an die Rückrufadresse der Workflowinstanz über SOAP/HTTP versendet. Dem Call-Objekt wurden dazu die Header für die WS-Addressing hinzugefügt und mit den Rückrufdaten des Workflows besetzt.
Der Deregistrierungsaufruf wird in ähnlicher Weise verarbeitet wie die Registrierung (s.o.). Dabei wird der EventInfoMapper verwendet, um ein EventInfo Dokument in ein EventInfo Objekt umzuwandeln. Danach wird im RegistrationHandler das Ereignis über die Schnittstelle zur Ereignisquelle deregistriert und die Rückrufdaten des Workflows werden aus dem Speicher entfernt.
In einer Konfigurationsdatei der Implementierung der Web Service Schnittstelle gibt es die Möglichkeit, die Adresse der Ereignisquelle und des Benachrichtigungsdiensts einzustellen. Diese Daten werden für die Erzeugung der Stubs zu den jeweiligen Diensten benötigt.
Für den Zugriff auf den Ereignisdienst wird für die Integrationsprozesse eine WSDL Beschreibung erzeugt, die in der Datei EventService.wsdl abgelegt wird (siehe Listing B.3).

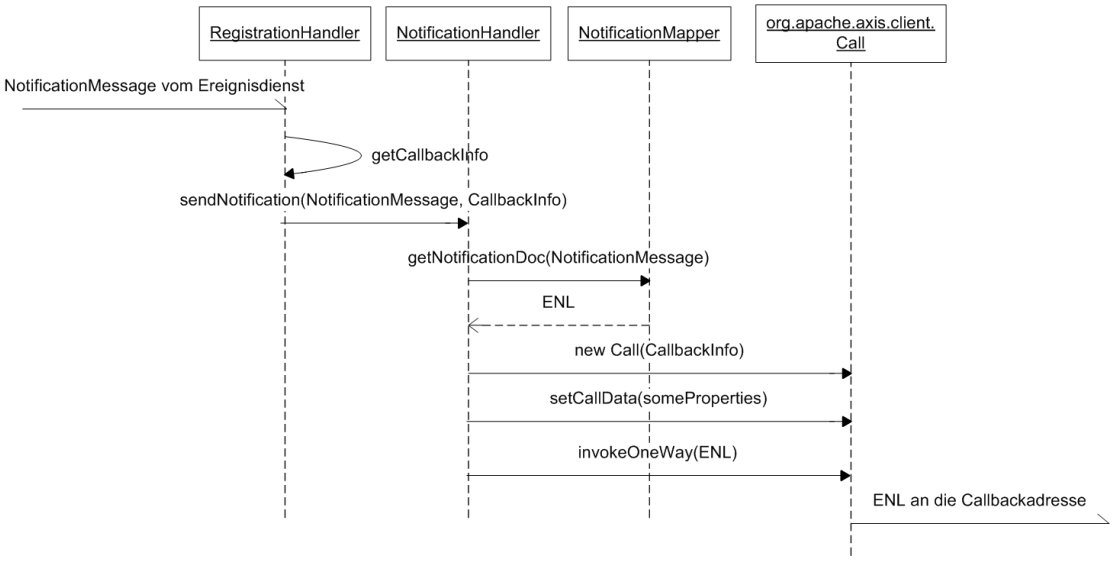

Abbildung 4.19: Sequenzdiagramm *Asynchrone Ereignisbenachrichtigung*

BPEL Prozesse

Die Integrationsprozesse ContextEvent und ContextEventOnEnterArea werden ebenfalls mit BPEL beschrieben. Für die Beschreibung der Prozesse wurden die wesentlichen Elemente bereits im Abschnitt 4.4.1 vorgestellt, deshalb sollen hier nur die zusätzlich erforderlichen Elemente aus der BPEL Spezifikation aufgeführt werden. Der Aufruf des Ereignisdiensts erfolgt bei Context-Event asynchron. Deshalb werden in einem partnerLink Element zwei Rollen definiert, von denen der Integrationsprozess die Rolle des EventServiceRequester, d.h. des Aufrufenden, und der Ereignisdienst die Rolle des EventServiceProvider, also dem Anbieter von Ereignisdiensten, erhält. Für den Empfang der asynchronen Benachrichtigung wird im Integrationsprozess das pick Element verwendet. Das pick Element besteht dabei aus zwei Elementen, dem onMessage Element und dem onAlarm Element. Im onMessage Element wird die asynchrone Operation receiveNotification implementiert. Dies geschieht in diesem Element durch die Verwendung der asynchronen Schnittstelle (EventServiceCallbackPortType) aus dem partnerLink Element des Ereignisdiensts. Sobald ein Rückruf über diese Schnittstelle erfolgt, wird er diesem Element zugewiesen. Die Zuweisung erfolgt in der BPEL Engine über WS-Addressing Header und ist für den Modellierer des Prozesses transparent. Die BPEL Engine sorgt dafür, dass der Prozesszweig bis zur Ankunft einer Benachrichtigung im Speicher abgelegt wird. Parallele Aktivitäten werden jedoch weiterhin von der BPEL Engine ausgeführt. Das onAlarm Element wird nach Ablauf der Gültigkeitsdauer eines Ereignisses aufgerufen.

XML Datentypen

Die Beschreibung der ausgetauschten Dokumente erfolgt mit XML Schema. Für die Dokumente ERL, EventInfo und ENL werden dafür bereits existierende Schemata verwendet. Diese sind in den jeweiligen Dateien EventRegistrationLanguage.xsd, EventInfo.xsd und EventNotification-Language.xsd definiert. Die Schemata werden unter der gleichen Adresse zur Verfügung gestellt wie in Abschnitt 4.4.1.

XSLT Stylesheets

Ein Stylesheet ist lediglich für den Integrationsprozess ContextEventOnEnterArea erforderlich. Es nimmt die Transformation des ContextEventOnEnterAreaRequest Formats (Quelldokument) in das Dokumentformat des ContextEvent Integrationsprozesses vor. In Listing D.2 ist das Stylesheet aufgeführt. Das Quelldokument wird im Vergleich zu Abschnitt 4.4.1 auf eine andere Weise verarbeitet. Dies veranschaulicht die vielen Möglichkeiten die XSLT bei der Transformation von Dokumenten bietet. Es wird hier nur ein einziges template Element für die Wurzel des Quelldokuments definiert. Danach werden die Werte aus dem Quelldokument mit XPath Ausdrücken ausgewählt und mit dem value-of Element der XSLT Spezifikation in das Zieldokument kopiert (hier ERL). Es wird z.B. der Wert thresholdProbability aus dem Quelldokument mit einer XPath Anweisung ausgelesen und in die ERL Struktur für thresholdProbability kopiert. Die Struktur des ERL Dokuments, in welches die Werte kopiert werden, wird dabei vom Stylesheet erzeugt. Viele Parameter sind zusätzlich für den Prozessmodellierer transparent, da sie vom Stylesheet gesetzt werden, z.B. blocking interval.

Pakete

Die BPEL Prozesse werden entsprechend Abschnitt 4.4.1 in ein Archiv gepackt und auf der BPEL Engine installiert. Die Web Service Schnittstelle zum Ereignisdienst besteht hauptsächlich aus den im Abschnitt 4.4.2 genannten Klassen. Die Klassen der Web Service Schnittstelle werden zusammen mit den Apache Axis Klassen in ein Web Archiv gepackt und in einem Web Container installiert. Für den Zugriff auf die Web Service Schnittstelle durch die BPEL Prozesse wird die WSDL-Beschreibung des Ereignisdiensts zusätzlich an einem zugreifbaren Ort zur Verfügung gestellt. In der WSDL Beschreibung wird vorher noch der Endpunkt eingetragen, an dem die Anfragen für den Ereignisdienst entgegengenommen werden.

4.4.3 GetMap

Die Schnittstelle GetMap hat viele Gemeinsamkeiten zu den Schnittstellen aus Abschnitt 4.4.1, besonders bezüglich der Realisierung der Web Service Schnittstelle zur Nexus Plattform. Die Abschnitte über die Realisierung der Web Service Schnittstelle, das Prozessmodell und die Datentypen basieren deshalb auf einer Beschreibung dieser Gemeinsamkeiten und Unterschiede.

Web Service Schnittstelle

Für den Kartendienst der Nexus Plattform existiert als Implementierung der Web Service Schnittstelle ebenfalls das SeifeServlet. Es nimmt Anfragen im MapPL Format entgegen und liefert Rückgaben im MapPL Format. Für die Dokumentformate existieren ebenfalls entsprechende Serialisierer und Deserialisierer (PayloadConverter). Für die Service-Klasse, die vom SeifeServlet aufgerufen wird, liefert der Kartendienst ebenfalls eine Implementierung. Die Verarbeitung der Anfragen entspricht dann exakt dem Ablauf aus dem Sequenzdiagramm in Abbildung 4.16, wobei als Eingabenachricht ein MapPL Dokument eintrifft und als Rückgabenachricht ein MapML Dokument gesendet wird. Das SeifeServlet erkennt am Format der gesendeten Nachricht, dass es die entsprechenden PayloadConverter und die entsprechende Serviceimplementierung des Kartendiensts verwenden muss. Für den Zugriff auf den Kartendienst wird für die Integrations-

prozesse eine WSDL Beschreibung erzeugt, die in der Datei MapService.wsdl abgelegt wird (siehe Listing B.1)

BPEL Prozesse

Für die Beschreibung der Prozessmodelle durch BPEL werden die selben Elemente verwendet, die bereits bei der ContextQuery Schnittstelle zum Einsatz kamen. In variable Elementen werden die MapPL und MapML Dokumente gespeichert. Für den Zugriff auf die Web Service Schnittstelle des Kartendiensts wird ein partner-Link definiert. Da der Kartendienst synchron aufgerufen wird, wird lediglich die Rolle MapServiceProvider für den Kartendienst spezifiziert, d.h. die Rolle des Anbieters von Kartendiensten. Die Struktur des BPEL Prozesses von GetMap ähnelt dem Prozess aus Listing C.1. In diesem Prozess müssten dann für den GetMap Integrationsprozess nur der Namensraum, das partnerLink Element, die Formate der Ein- und Ausgabevariablen und der Aufruf im invoke Element angepasst werden.

XML Datentypen

Für die Formate MapPL und MapML werden bereits existierende Schemata verwendet. Diese sind in den jeweiligen Dateien NexusMapMLSchema.xsd und NexusMapPLSchema.xsd definiert. Sie werden unter der gleichen Adresse zur Verfügung gestellt wie in Abschnitt 4.4.1.

XSLT Stylesheets

Im GetMap Integrationsprozess wird keine Transformation an Dokumenten vorgenommen. Ein verfeinerter Integrationsprozess, der zukünftig auf GetMap aufbaut und dafür eine Transformation definiert, ist denkbar (siehe Abschnitt 5.3).

Pakete

Der BPEL Prozess wird wie in den vorigen Abschnitten auf der BPEL Engine installiert. Die WSDL-Beschreibung des Kartendiensts (MapService.wsdl) wird an einem zugreifbaren Ort zur Verfügung gestellt. In der WSDL Beschreibung wird vorher noch der Endpunkt eingetragen, an dem das SeifeServlet die Anfragen für den Kartendienst entgegennimmt.

4.5 Inbetriebnahme

In Abbildung 4.20 sehen wir die Verteilung der Komponenten in einer Testumgebung. Auf dem Testserver wird ein Oracle BPEL Server installiert. Auf dem BPEL Server befinden sich sowohl die Integrationsprozesse als auch eine Workflowanwendung (siehe Abschnitt 4.6), die über die Integrationsprozesse auf die Nexusdienste zugreift. Die Integrationsprozesse nutzen die Web Service Schnittstellen der Nexus Plattform (Seife Servlet und Event Axis WS). Die Kommunikation mit der Nexus Plattform durch die Workflowanwendung erfolgt ausschließlich über SOAP/HTTP im Document/Literal Nachrichtenstil. Die Föderation und der Ereignisdienst sind zusammen mit dem SeifeServlet bzw. der Event Web Service Schnittstelle im gleichen Web Container installiert (hier Apache Tomcat), könnten jedoch auch auf unterschiedlichen

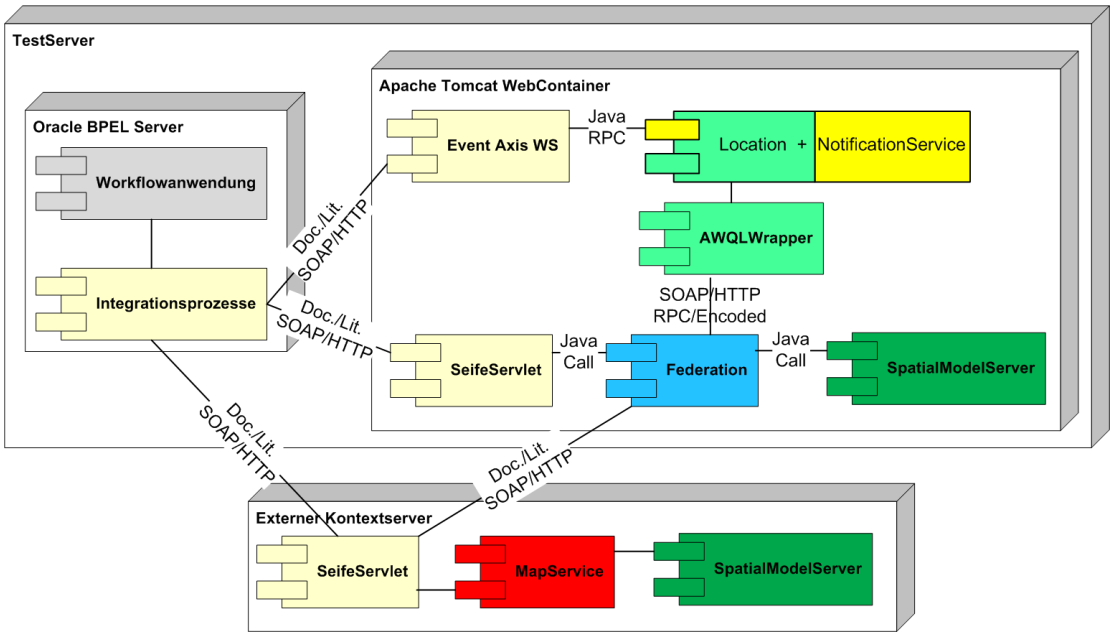

Abbildung 4.20: Testumgebung der Implementierung

Servern eingerichtet sein (siehe Kartendienst). Der Lokationsdienst (Location Service) und der Benachrichtigungsdienst (Notification Service) werden in der Testumgebung als Bundle installiert, da bislang nur vom Lokationsdienst die Funktionalität einer Ereignisquelle implementiert wird (siehe Abschnitt 2.1.5). Der Lokationsdienst stellt dabei zusätzlich, in seiner Funktion als Kontextserver, Informationen über bewegliche Objekte zur Verfügung (siehe Abschnitt 2.1.3). Die Föderationskomponente sorgt für die Verteilung der Anfragen an den SpatialModelServer und die Server des Lokationsdiensts über das SeifeServlet bzw. den AWQLWrapper. Der Kartendienst ist bereits als Dienst auf einem Server des Nexus Projekts installiert (Externer Kontextserver). Der Kartendienst wird von den Integrationsprozessen über das SeifeServlet angesprochen.

4.6 Beispielprozess „Paketdienst”

Für die Evaluierung der *Integrationsprozesse* und somit der Integration von Kontextinformationen in Workflows, wird in diesem Abschnitt eine Beispielanwendung vorgestellt. Das Szenario für die Anwendung ist folgendes: In einem Unternehmen gibt es verschiedene Arbeitsbereiche, z.B. die Produktion, das Lager oder die Verwaltung. Für den Austausch von nicht elektronischen Informationen zwischen den Abteilungen, z.B. von Lieferscheinen oder Ersatzteilen, müssen Pakete von einer Abteilung zu einer anderen gebracht werden. Die Abholung und Auslieferung der Pakete soll dabei durch einen Boten erfolgen.
Die informationstechnische Unterstützung des Abhol- und Auslieferungsprozesses aus dem Szenario wird dabei durch ein Workflowsystem unterstützt. Dafür wird ein Workflow modelliert, der in einem ersten Schritt ein vorhandenes Paket in einem bestimmten Gebiet sucht. Für diese kontextbezogene Suche wird der *ContextQuery Integrationsprozess* verwendet. Sobald ein Paket gefunden wird, wird ebenfalls über den *ContextQuery Integrationsprozess* ein Bote gesucht, der sich in der näheren Umgebung des Pakets befindet. Dem Boten wird daraufhin die Aufgabe zugeteilt, das Paket abzuholen. Die Aufgabenzuteilung kann auf verschiedene Arten erfolgen,

z.B. durch den Versand einer SMS (Short Message Service) an das Mobiltelefon des Boten oder durch die Bereitstellung der Aufgaben in einem zusätzlichen Dienst, der vom Boten abgefragt werden kann (s.u. TaskManagerService). Nachdem der Bote den Auftrag angenommen hat, wird der Workflow die Abholung der Lieferung durch den Boten überwachen. Zur Überwachung wird ein Ereignis über den *Integrationsprozess ContextEventOnEnterArea* angemeldet, das den Workflow benachrichtigt, sobald das Paket abgeholt wurde. Schließlich wird die Auslieferung des Pakets am Zielort überwacht.
In den nachfolgenden Abschnitten wird das Prozessmodell der Workflowanwendung „Parcel-Service" im Detail vorgestellt. Zuvor werden jedoch die Boten und die Pakete des Lieferdiensts in der Augmented World modelliert. Für die Simulation der Objektbewegungen wird zum Schluss eine grafische Oberfläche vorgestellt.

4.6.1 Modellierung von Boten und Lieferungen

Für die Workflowanwendung, die das Szenario des Lieferdiensts aufgreifen soll, werden Boten und Pakete in der Augmented World modelliert. Für beide werden erweiterte Schemata erstellt (siehe Anhang A.1). Dabei werden die Klassen *Postman* und *Parcel* in einem erweiterten Klassenschema definiert. Beide Klassen basieren auf der Klasse *MobileObject* aus dem *NSCS*. Parcel und Postman haben zusätzliche Attribute, die sie von anderen mobilen Objekten unterscheiden. Bei Parcel sind es Eigenschaften wie die Paketnummer, der Abholort, der Lieferort, der Empfänger, das Gewicht oder die Lieferzeit. Bei Postman ist es die Personalnummer oder das maximale Gewicht, das getragen werden kann. Alle erweiterten Attribute basieren dabei auf den Standardattributen aus dem *NSAS* und werden in einem erweiterten Attributschema definiert. Die Informationen dieser Eigenschaften werden von der Workflowanwendung genutzt.

4.6.2 Workflow für den Lieferdienst - *ParcelService*

In Abbildung 4.21 sehen wir den Workflow für den Lieferdienst. Der Workflow stellt seinen Dienst über eine WSDL Beschreibung zur Verfügung und definiert für die Kommunikation ein eigenes Nachrichtenformat (ParcelServiceRequest). Das Nachrichtenformat ParcelService-Request ist für die Beispielanwendung bewusst einfach gewählt worden: Es wird die Definition eines Gebiets anhand von zwei gegenüberliegenden geografischen Punkten (links oben und rechts unten) erwartet und die Angabe der Laufzeit des Lieferdienstprozesses in Minuten. Mit diesen Angaben wird der Prozess in der angegebenen Zeit versuchen, die Sendungen im spezifizierten Gebiet aufzufinden und auszuliefern. Die Koordinaten werden dabei durch den Längen- und Breitengrad im Format WGS84 angegeben (siehe Anhang A.2).
Mit den Koordinaten aus dieser Nachricht wird der asynchrone Unterprozess *PickupParcel* aufgerufen, der für die Abholung eines Pakets zuständig ist. Auf das Ergebnis von PickupParcel wird im pick Element gewartet. Trifft nach Ablauf einer festgelegten Zeit keine Nachricht ein, so wird abgebrochen und ein neuer Suchvorgang gestartet. Trifft eine Nachricht ein, so werden die darin vorkommenden Daten für die Zustellung des Pakets durch den Prozess *DeliverParcel* verwendet. Sobald das Paket ausgeliefert worden ist, wird dies dem aufrufenden Klienten-Prozess mitgeteilt. Ansonsten wird der gesamte Prozess neu gestartet.

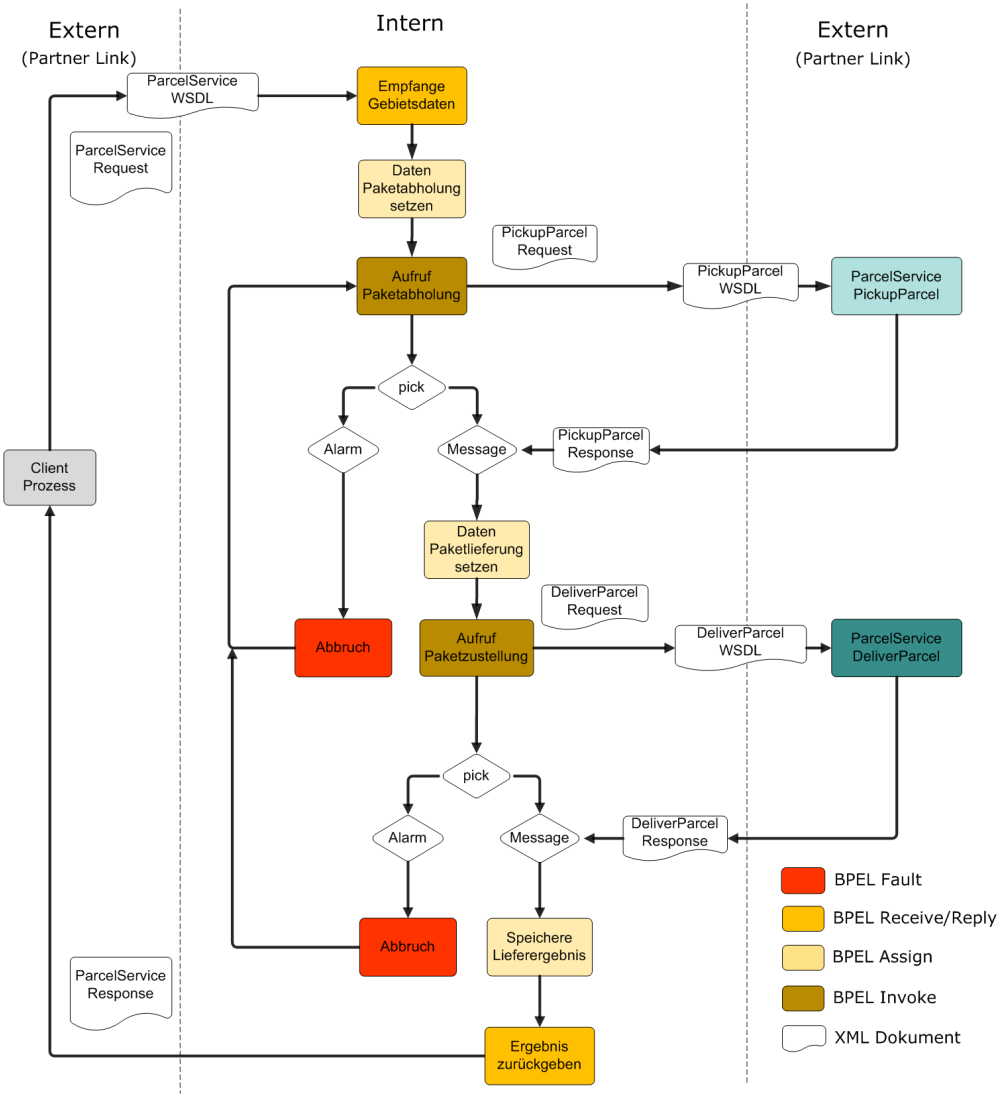

Abbildung 4.21: Workflowmodell Lieferdienst

Teilprozess zum Abholen von Paketen - *PickupProcess*

In Abbildung 4.22 sehen wird das Prozessmodell von PickupParcel. Der Prozess stellt seinen Dienst ebenfalls über eine WSDL Beschreibung zur Verfügung. Auch hier wird ein neues Dokumentformat definiert, das von der aufrufenden Anwendung eingehalten werden muss. Folgende Elemente werden erwartet:

- Ein String, in dem ein Gebiet, beschrieben durch WGS84 Koordinaten, übergeben wird, sowie
- ein Integer Wert, in dem die maximale Dauer für die Abholung eingetragen wird.

Mit diesen Informationen wird zuerst eine Anfrage formuliert, mit der ein Paket im angegebenen Gebiet gesucht werden sollen. Diese Anfrage wird an den Integrationsprozess ContextQuery gerichtet (siehe Abschnitt 4.3.1). In einem vorgelagerten Transformationsschritt wird dafür das Anfragedokument für ContextQuery erstellt. Die Antwort von ContextQuery enthält

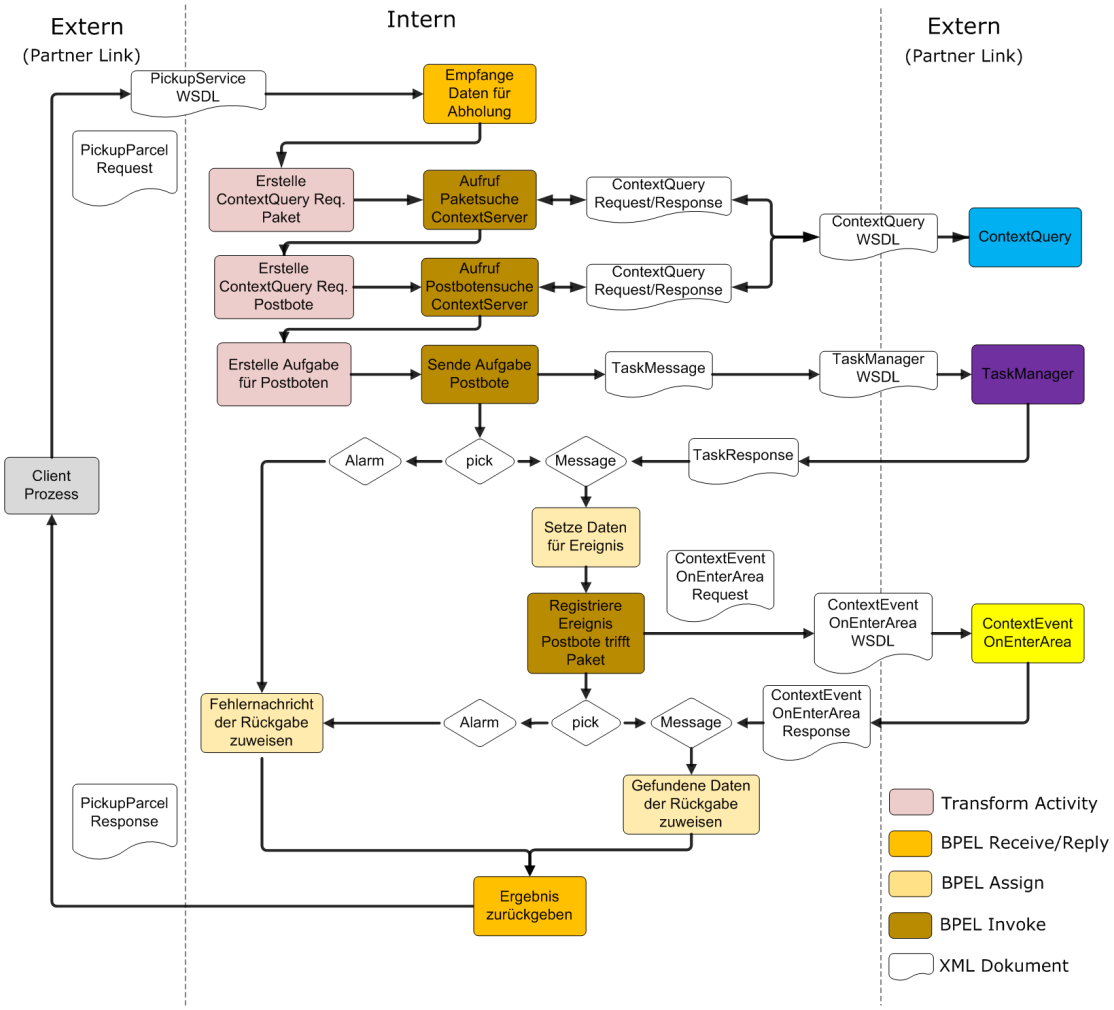

Abbildung 4.22: Workflowmodell PickupParcel

Informationen über Pakete in der Umgebung. **Für die Extraktion der Paketdaten verwenden wir erstmals einen Transformationsschritt, der AWML Nachrichten verarbeitet**. In Listing 4.16 ist ein Auszug aus dem XSLT Stylesheet der Transformation zu sehen. Im Stylesheet werden für sämtliche Attribute, die für die Workflowanwendung von Interesse sind, match-Anweisungen angegeben:

- Mit *parcelId* wird nach dem Bezeichner für Pakete gesucht.
- Mit *receiverPoint* wird der Lieferort des Pakets ermittelt.
- Mit *name* wird die Paketbezeichnung ermittelt.

Der XSLT Prozessor kopiert diese Werte in eine Variable des Prozesses (Zieldokument), welche für die Speicherung der Paketdaten definiert worden ist. Im Zieldokument sind die Attribute nicht mehr generisch definiert, sondern vom jeweiligen Typ des Attributs, d.h. für parcelId ist dort, anstatt des generischen Elements NexusAttribute, das spezielle Element parcelId aus dem ParcelDeliveryAttributeSchema angegeben. Somit kann in späteren Prozessaktivitäten über eine XPath Anweisung direkt auf die Attribute zugegriffen werden. Das Problem der unbekannten generischen Typen wird dadurch aufgelöst.

Listing 4.16: Stylesheet zur Verarbeitung von Paketdaten

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:awql="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
 xmlns:parcelecs="http://www.nexus.uni-stuttgart.de/ParcelDeliveryExtendedClassSchema"
 xmlns:parcelas="http://www.nexus.uni-stuttgart.de/ParcelDeliveryAttributeSchema"
 xmlns:client="http://nexus.informatik.uni-stuttgart.de/ContextAwareProcesses/ParcelServiceAreaAssignTask"
 ...(Weitere Namensräume von Nexus)...>
 <xsl:template match="/">
   <client:ParcelData>
     <xsl:apply-templates/>
      </client:ParcelData>
 </xsl:template>
 <xsl:template match="*">
  <!-- Elements which are not handled by match functions will be ignored-->
 </xsl:template>
 <xsl:template match="awml:awml">
      <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="awml:nexusobject">
      <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="parcelas:parcelId">
      <xsl:if test="*">
          <parcelas:parcelId>
              <xsl:copy-of select="*"/>
          </parcelas:parcelId>
        </xsl:if>
 </xsl:template>
 <xsl:template match="parcelas:receiverPoint">
      <xsl:if test="*">
          <parcelas:receiverPoint>
              <xsl:copy-of select="*"/>
          </parcelas:receiverPoint>
        </xsl:if>
 </xsl:template>

  ...(Weitere Attribute die verarbeitet werden sollen)...

 <xsl:template match="nsas:name">
      <xsl:if test="*">
          <nsas:name>
              <xsl:copy-of select="*"/>
          </nsas:name>
        </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Mit den ermittelten Paketdaten wird anschließend nach einem Boten in der Umgebung des Pakets gesucht. Deshalb wird in einem zweiten Transformationsschritt erneut eine Anfrage im AWQL Format an die ContextQuery Schnittstelle formuliert. Die Rückgabe der ContextQuery Schnittstelle liefert uns dann eine AWML Nachricht mit den Daten eines Boten. Die Verarbeitung der AWML Nachricht entspricht der Vorgehensweise bei den Paketdaten. Nachdem sowohl ein Paket als auch ein Postbote in der Nähe des Pakets ermittelt werden konnten, erfolgt die Vergabe der Aufgabe an den Postboten. Dafür wird an den *TaskManagerService*, der bereits in der BPEL Engine von Oracle integriert ist, ein *Task* gesendet. Ein Task steht dabei stellvertretend für eine Aufgabe bzw. einen Auftrag. Deshalb werden im Task die Paketdaten für den Postboten abgelegt, die er zur Abholung des Pakets benötigt. Der PickupProcess wartet anschließend darauf, dass die Aufgabe vom Postboten übernommen wird. Sobald der Bote den Auftrag annimmt, z.B. indem er es aus einer Liste auf seinem tragbaren Computer auswählt, wird der TaskManagerService eine Bestätigung an den PickupProcess senden. Der PickupProcess wartet auf die asynchrone Auftragsbestätigung in einem pick Element.
Nachdem die Bestätigung eingetroffen ist, ist die Zuordnung des Postboten zum Paket erfolgt. In einem nächsten Schritt registriert der Workflow über den Integrationsprozess ContextEvent-OnEnterArea das Ereignis der Paketabholung (siehe Abschnitt 4.3.4). Sobald der Postbote in die unmittelbare Nähe des Pakets kommt, z.B. bei einer Entfernung von fünf Metern, wird ein Ereignis ausgelöst. Für den Aufruf von ContextEventOnEnterArea werden die bereits vorhandenen

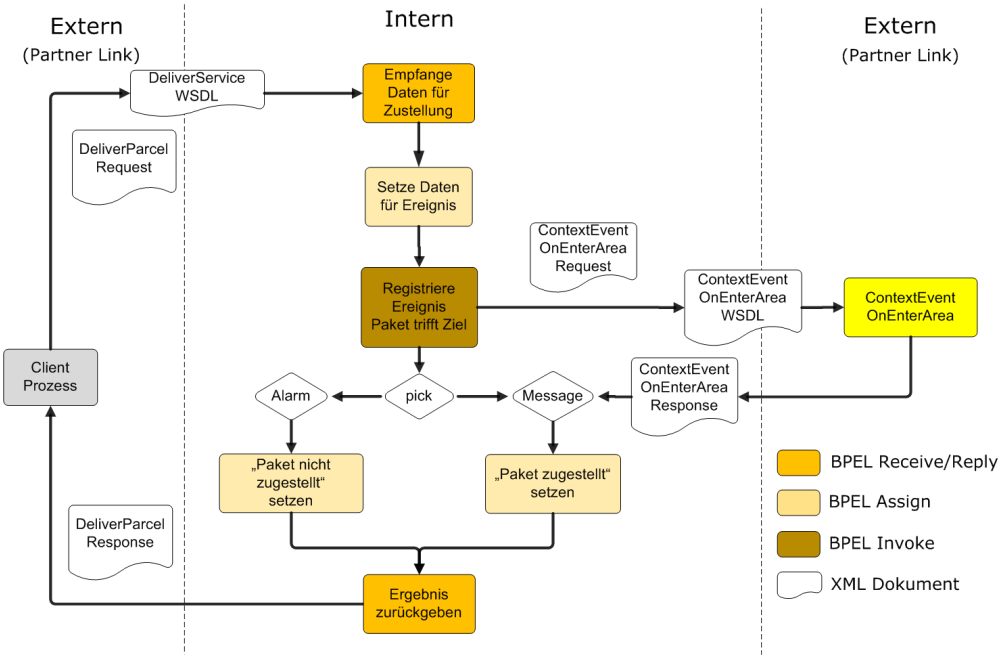

Abbildung 4.23: Workflowmodell DeliverParcel

Paket- und Botendaten verwendet. Da der Integrationsprozess eine einfachere Schnittstelle zur Verfügung stellt, ist hier keine Transformation nötig. Nach Eintreten des Ereignisses, d.h. der Postbote hat das Paket entgegengenommen, wird das Ergebnis der Abholung an den übergeordneten Prozess (ParcelService) weitergegeben.

Teilprozess zur Auslieferung von Paketen - *DeliverProcess*

In Abbildung 4.23 ist schließlich der Workflow für die Auslieferung des Pakets dargestellt. Hierbei handelt es sich im Wesentlichen um die Registrierung eines Ereignisses beim Ereignisdienst, das ausgelöst wird, sobald das Paket am Zielort ausgeliefert wird. Dafür wird wieder ein Ereignis vom Typ On Enter Area verwendet. Zu beachten ist, dass hier das Paket den Zielort erreichen muss, d.h. es wird die Bewegung des Pakets überwacht. Im PickupProcess hingegen wurde die Bewegung des Boten überwacht. Somit ist gewährleistet, dass das Paket auch wirklich den Lieferort erreicht.

4.6.3 Simulation der Bewegung von Boten und Lieferungen

Für die Simulation der Bewegung von mobilen Objekten in der Augmented World wurde der EventDemoClient aus dem Nexus Projekt verwendet. Im Hauptfenster des EventDemoClient ist ein zweidimensionaler Kartenausschnitt eines Gebiets dargestellt, in dem sich die Objekte der Augmented World befinden (siehe Abb. 4.24). Der Kartenausschnitt wird dabei vom Kartendienst der Nexus Plattform erzeugt. Die Funktionalität des EventDemoClient wurde erweitert, so dass über zwei Schaltknöpfe jeweils ein Postbote und ein Paket erzeugt werden können. Die Platzierung dieser Objekte auf dem Kartenausschnitt simuliert das Vorhandensein der Objekte in der Augmented World. Die Platzierung erfolgt über die Computermaus. Das entsprechende

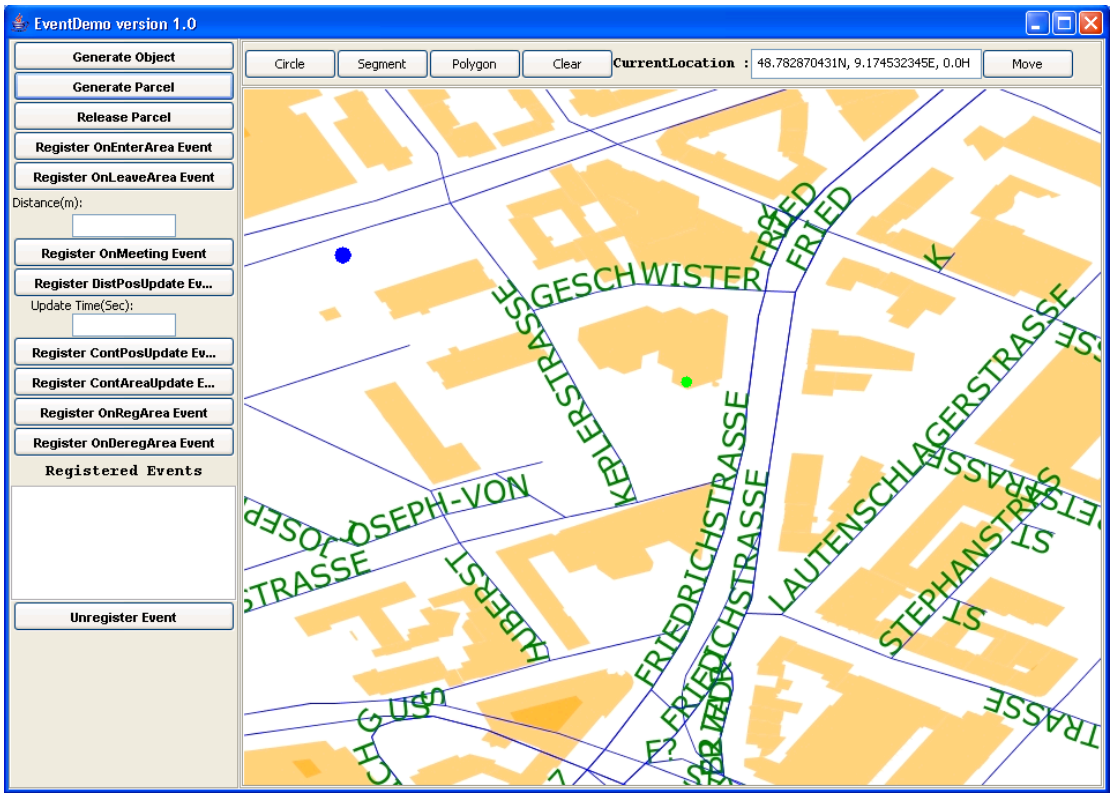

Abbildung 4.24: Anwendungsoberfläche zum Simulieren der Objektbewegung

Symbol für den Postboten ist ein großer blauer Punkt und für das Paket ein kleiner grüner Punkt. Durch Klicken mit der Computermaus in den Kartenausschnitt kann der Postbote (blauer Punkt) um das Gebiet des Pakets (grüner Punkt) bewegt werden, so wie auch in der Realität sich ein Bote im Liefergebiet fortbewegt. Für den Einsatz der Anwendung in einem realen Szenario müssten die Boten und Lieferungen mit entsprechenden Sensoren ausgestattet werden (z.B. GPS-Empfänger), um deren Bewegung verfolgen zu können.

5 Bewertung der Ergebnisse

In diesem Kapitel sollen die Ergebnisse dieser Arbeit bewertet werden. Dazu wird zuerst eine Bewertung der funktionalen und nichtfunktionalen Anforderungen durchgeführt, die zu Beginn des Entwurfs aufgestellt wurden. In einem eigenen Abschnitt werden die Einschränkungen des im Hauptteil erarbeiteten Konzepts aufgeführt. Anschließend werden Ideen für mögliche Erweiterungen der Arbeit vorgestellt. Schließlich werden im letzten Abschnitt einige Regeln und Hinweise formuliert, die nützlich für die Modellierung von kontextbasierten Workflows sind und die auf den bisher gemachten Erfahrungen bei der Erstellung von Workflowanwendungen basieren.

5.1 Auswertung der Anforderungen

Für die wesentlichen Nexusdienste Föderation, Ereignisdienst und Kartendienst, wurden in dieser Arbeit Integrationsprozesse erstellt, so dass die Funktionen der Nexus Plattform auch im Workflow zur Verfügung stehen.
Die Integrationsprozesse der ersten Ebene (siehe Abschnitt 4.2.1) halten dabei für die Kommunikation mit den Nexusdiensten die komplexen Nachrichtenformate von Nexus ein (siehe Abschnitt 4.1.1). Hierdurch ist gewährleistet, dass die Funktionalität der Nexus Plattform vollständig genutzt werden.
Durch die funktionale Einteilung der Integrationsprozesse auf der ersten Ebene ist für den Modellierer zudem klarer, welcher Integrationsprozess für das Abfragen, das Ändern und das Modellieren von Kontextdaten, die Registrierung von Ereignissen und die Generierung von Karten zuständig ist. Welcher konkrete Nexusdienst dagegen für die Ausführung der Anfragen genutzt wird, ist für den Modellierer unwesentlich und bleibt transparent. Der entsprechende Nexusdienst kann aber bei Bedarf durch einen Blick in die Prozessbeschreibung des Integrationsprozesses ermittelt werden.
Durch die Erweiterungsmöglichkeit der Integrationsprozesse der ersten Ebene lassen sich neben der Bereitstellung der vollen Funktionalität von Nexus zusätzlich vereinfachte oder anwendungsspezifische Integrationsprozesse entwickeln. Dadurch ist die einfache Integration der Nexusdienste gewährleistet (siehe ContextEventOnEnterArea).
Bei Änderungen an den Nexusdiensten ist nur dann eine Änderung an den Integrationsprozessen notwendig, wenn sich die Beschreibung der Nexus Schnittstellen (Nachrichtenformate) ändert. Diese Änderung ist jedoch ein gravierender Schritt, der sich nicht nur auf die Integrationsprozesse und auf Workflowanwendungen auswirken würde, sondern auf sämtliche Anwendungen, die auf diese Nexus Schnittstelle vertrauen. Der Vorteil in der Hierarchie der Integrationsprozesse ist aber, dass eventuelle Änderungen bereits auf der ersten oder zweiten Ebene abgefangen werden können und den verfeinerten Integrationsprozessen höherer Ebene erspart bleiben, da sie eventuell nicht mehr an das Nachrichtenformat von Nexus gebunden sind. Folgendes Beispiel: Ändert sich das Format für die Registrierung von Ereignissen, so muss im ContextEvent Integrationsprozess das neue Nachrichtenformat eingebunden werden und im ContextEventOnEnterArea

die Transformation in das neue Nachrichtenformat abgeändert werden. Am DeliverProcess der Beispielanwendung, der den Integrationsprozess und somit das Nachrichtenformat von ContextEventOnEnterArea verwendet, muss keine Änderung mehr durchgeführt werden (siehe Abschnitt 4.6.2). Entwicklungsumgebungen können zur schnellen Durchführung von Änderungen und einer entsprechenden Neuinstallation der Prozesse hilfreich sein. Falls sich der Endpunkt für die Bereitstellung des Nexusdiensts ändert, so erfordert das nur eine Anpassung des Endpunkts in der WSDL Beschreibung.
Die Modellierung von Workflowanwendungen ist einfacher und intuitiver im Vergleich zur klassischen Programmierung, solange der Anwendungsbereich prozessorientiert ist, z.B. ein Fertigungsprozess oder ein Abhol- und Lieferprozess. Zudem wird durch die Prozessorientierung auch die Beteiligung der Kunden oder der zukünftigen Benutzer des Systems, die Teil des Geschäftsprozesses sind, bei der Entwicklung der Anwendung verbessert. Durch unterstützende Funktionen einer geeigneten Entwicklungsumgebung, wie z.B. einem grafischen Editor zur Anfertigung von Prozessmodellen, wird die Modellierung zusätzlich unterstützt. Die Entwicklung wirkt leichter, robuster und führt schneller zum Erfolg. Die Konzentration liegt auf der Modellierung der Prozesslogik und weniger auf der Beseitigung von Fehlern in der Programmsyntax. Nach einer Einarbeitung in die wesentlichen Technologien bei der Workflowmodellierung (siehe Kapitel 2) und bei der Beachtung anfänglicher Regeln und Hinweise (siehe Abschnitt 5.4) ist die Entwicklung von nützlichen Workflowanwendungen, die Kontextinformationen verarbeiten, in wenigen Tagen möglich. Die Akzeptanz der Workflowmodellierung gegenüber der klassischen Programmierung bei der Entwicklung kontextsensitiver Anwendungen, sowie der Aufwand und die Bearbeitungszeit müssen anhand zahlreicher Beispielanwendungen (siehe Abschnitt 4.6) evaluiert werden.

5.2 Einschränkungen

Bei der Integration von Workflowanwendungen in die Nexus Plattform gibt es gewisse Einschränkungen in folgenden Bereichen:

Nexusdienste generell Die Entwicklung einer Plattform für kontextbasierte Anwendungen ist komplex und ständig evolutionären Schritten ausgesetzt, die zur Ausweitung der Funktionalität und der Strukturen führen. Deshalb kann es bisweilen vorkommen, dass bestimmte Funktionen nicht implementiert sind bzw. beim Übergang von einer Entwicklungsstufe in die nächste noch nicht übernommen wurden. So ist z.B. beim Einfügen von neuen Attributen in einem append Element einer AWQL Anfrage nur die Angabe eines Attributs möglich oder bei der Auswahl von Attributen in einem include Element einer AWQL Anfrage hat die Angabe eines zusätzlichen restriction Elements keine Auswirkung. Die Funktionalität eines Diensts kann im Vorfeld über eine Anfrage mit einem separaten Programm (z.B. [soapUI]) getestet werden.

AWQLWrapper Der Location Service stellt Kontextinformationen mobiler Objekte bereit. Der AWQL Wrapper des *Location Services* bietet deshalb hauptsächlich eine Wrapperfunktion für Anfragen an, die sich auf Eigenschaften mobiler Objekte beschränken, also z.B. die Position oder die Geschwindigkeit. Für die Beispielanwendung aus Abschnitt 4.6 wurde der AWQL Wrapper so erweitert, dass er zusätzlich Objekte und Eigenschaften aus dem erweiterten Schema für den Lieferdienst kennt, z.B. die Klasse *Parcel* für das Paket und das Attribut *receiverPoint* für den Lieferort. Die Realisierung zusätzlicher Attribute und Klassen ist im AWQL Wrapper momentan zu aufwendig. Für zukünftige Anwendungen,

die zusätzliche Kontextinformationen zu mobilen Objekten benötigen, muss der Location Service um die Funktionalität eines Spatial Model Servers erweitert werden bzw. der Spatial Model Server um die Funktionalität des Location Service.

Föderation Bislang existiert keine Anbindung des Location Servers an die Föderation, so dass mobile Objekte nicht über die Föderation abgefragt werden können. Das Problem der Anbindung besteht darin, dass der Location Server über den AWQLWrapper Nachrichten im Stil RPC/Encoded akzeptiert, die Föderation jedoch Nachrichten im Stil Document/-Literal versendet. Die Schnittstelle des Location Servers kann folglich die Nachrichten der Föderation nicht verarbeiten. Für die Beispielanwendung wurde jedoch die Föderation so abgeändert, dass Anfragen, die an den Location Service gerichtet sind, vorher in den RPC/Encoded Stil umgewandelt werden. Die Umwandlung geschieht momentan in der gleichen Klasse, die von der Föderation für den Versand von Nachrichten im Document/-Literal Stil verwendet wird. In Zukunft ist eventuell die Implementierung in einer separaten Klasse oder die Anpassung der Schnittstelle des Location Service erforderlich.

BPEL Engine Die Integrationsprozesse und die Beispielanwendung entstanden auf der Entwicklungsumgebung JDeveloper von Oracle und die Installation der Prozesse erfolgte auf dem Oracle BPEL Server. Da die Entwicklungsumgebung zum Teil Funktionen der BPEL Engine ausnützt, wie z.B. den Aufruf eines XSLT Prozessors, sind für die Migration der Prozesse auf andere Umgebungen kleine Anpassungen in den Prozessbeschreibungen der Integrationsprozesse nötig. Bei der Arbeit mit dem BPEL Server in der Version 10.1.2 konnte außerdem festgestellt werden, dass Änderungen an XSLT Stylesheets bei einer Neuinstallation des Prozesses nicht übernommen wurden. Hierfür war entweder ein Neustart des BPEL Servers oder die Umbenennung des Stylesheets erforderlich.

5.3 Mögliche Erweiterungen

In diesem Abschnitt sollen mögliche Erweiterungen vorgestellt werden, die im Rahmen dieser Arbeit fortgeführt werden können. Dabei wird die Definition neuer Schnittstellen vorgeschlagen, die Bereitstellung der Nexusdienste diskutiert, die Forderung nach speziellen Aktivitäten in Workflows aufgestellt und die Verbindung verschiedener Softwareprodukte über einen *Enterprise Service Bus* erörtert.

5.3.1 Definition zusätzlicher Schnittstellen

In dieser Arbeit wurden einige Integrationsprozesse entworfen, mit denen Workflowanwendungen auf Kontextinformationen und weitere Dienste der Nexus Plattform zugreifen können. Die vorhandenen Integrationsprozesse sind hierarchisch aufgebaut und können für zukünftige Anwendungen erweitert oder angepasst werden. So kann die Funktionalität auf der zweiten Ebene der Integrationsprozesse erweitert werden, indem beispielsweise neben ContextEventOnEnterArea weitere Integrationsprozesse für die übrigen Ereignisarten, wie z.B. Distance Position Update oder On Meeting, basierend auf ContextEvent entworfen werden. Ein anderer Grund für die Entwicklung eines verfeinerten Integrationsprozess wäre die Erfüllung der Forderung nach einer Schnittstelle, die wenig Eingabedaten erwartet, dafür aber eine ganz spezielle Funktion erfüllt, was z.B. vom Integrationsprozess ContextEventOnEnterArea erfüllt wird, der nur für die Beobachtung von On Enter Area Ereignissen genutzt werden soll. Ein weiteres Argument für eine Verfeinerung wäre das Ausblenden oder automatische Setzen von Parametern. In einem

verfeinerten Integrationsprozess, der auf der GetMap Schnittstelle basiert, kann z.B. ein spezielles Kartendesign realisiert werden, in dem die Zeichenregeln, das Format und die CSS Formatierungen vorgegeben sind. Für die Generierung einer Karte müsste dann lediglich das Gebiet angegeben werden. Ein letzter und wesentlicher Grund für die Erstellung von zusätzlichen Integrationsprozessen ist die Anpassung an bestimmte Anwendungsbereiche. Die Nexus Plattform ist bewusst generisch gehalten, um verschiedenen Anwendungen die Integration zu ermöglichen. Die Integrationsprozesse unterstreichen diese Eigenschaft und bieten durch ihre Erweiterungsmöglichkeit verschiedenen Anwendungen die Möglichkeit, auf Kontextinformationen zuzugreifen und zusätzliche Integrationsprozesse zu definieren, die auf ihren Bereich zugeschnitten sind. So wäre es für den Lieferdienst aus der Beispielanwendung (siehe Abschnitt 4.6) denkbar, verfeinerte Integrationsprozesse bereitzustellen, die eine Abfrage von Paket- und Botendaten auf Basis von ContextQuery erleichtern würden.

5.3.2 Bereitstellung von Diensten

Bislang werden die WSDL Beschreibungen der Nexusdienste auf einem Webserver abgelegt. Diese WSDL Beschreibungen könnten in einer Erweiterung gemäß der Rollenverteilung bei einer serviceorientierten Architektur (siehe Abschnitt 2.3) in einem Dienstverzeichnis abgelegt werden.
Des Weiteren kann die Einbindung eines Nexusdiensts zur Laufzeit des Workflows erfolgen. Dies bietet sich an, wenn mehrere Dienste die gleiche Funktionalität anbieten, z.B. wenn mehrere Instanzen des Föderationsdiensts existieren. Der Workflow lässt sich dann durch Parametrisierung so erweitern, dass zur Laufzeit eine dieser Instanzen über die WSDL Beschreibung, die im Verzeichnisdienst abgelegt ist, gefunden und gebunden wird. Ein Ansatz dafür wird bei [Wut06] beschrieben.
Ein anderer Ansatz für den Zugriff auf Dienste wäre, die Nexus Plattform für die Ablage von Dienstbeschreibungen zu verwenden. Dabei werden die Dienstbeschreibungen an bestimmten Orten der Augmented World abgelegt. Folgendes Szenario: Eine Person läuft mit einem tragbaren Computer (PDA oder Smartphone) an einem Kino vorbei. Das Kino bietet seine Dienste, z.B. den Abruf des aktuellen Kinoplans oder die Reservierung von Filmen, über eine Web Service Schnittstelle an. Dafür hat sie eine WSDL-Beschreibung der Dienste in ihrer unmittelbaren Umgebung abgelegt. Der tragbare Computer bindet den Dienst ein und reserviert einen Film. Für das Ablegen von Diensten in der Augmented World müsste eine Modellierung dieser Dienste auf der Nexus Plattform erfolgen, z.B. in Form eines erweiterten Schemas, das die wesentlichen Elemente einer WSDL Beschreibung enthält. Für das Einbinden der Dienste müssten die Schnittstellen entweder in gewisser Weise standardisiert werden, so dass z.B. jedes Kino die gleichen Funktionen für die Buchung und das Abrufen des Kinoplans anbietet oder die Schnittstellen und Funktionen müssten mit semantischen Mitteln beschrieben werden, so dass die Anwendung aufgrund der semantischen Beschreibungen die richtige Funktion für den Abruf des Kinoplans wählt.

5.3.3 Definition von Kontextaktivitäten in BPEL

Für den Abruf von Kontextinformationen bietet es sich an, spezielle Aktivitäten in einer Workflowmaschine zu definieren. Hierfür könnte die BPEL Spezifikation um kontextsensitive Bestandteile erweitert werden. Mit einer *invokeNearService* Aktivität, die eine normale invoke Aktivität erweitert, könnten beispielsweise ausschließlich Dienste in räumlicher Nähe angespro-

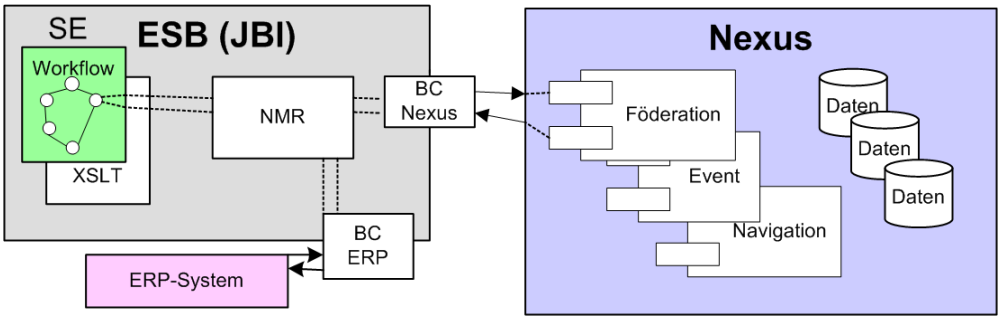

Abbildung 5.1: Nexusdienste in einem ESB

chen werden.
Für die Schnittstellen zur Nexus Plattform könnten die in dieser Arbeit definierten Integrationsprozesse ebenfalls als neue Aktivitäten eines Workflows definiert werden. So werden die Nexusdienste nicht über partnerLink Elemente aufgerufen, sondern über die neu definierten *Kontextaktivitäten*. Ein Ansatz dazu wird in [WKNL07] vorgestellt.

5.3.4 Erweiterung der Architektur

Für die Adressierung von Diensten wurde bei der Ereignisverarbeitung die WS-Addressing Spezifikation aus der Web Service Architektur verwendet (siehe Abschnitt 2.3). Es gibt weitere sinnvolle Bestandteile der Web Service Architektur, die zusätzlich bei der Nutzung der Nexus Plattform verwendet werden können. So kann beispielsweise die WS-Security Spezifikation für einen sicheren Datenaustausch implementiert werden. Die WS-Policy Spezifikation kann für die Aushandlung von Quality of Service (QoS) Parametern oder von zu verwendenden Sicherheitsmechanismen verwendet werden. Diese Funktionalitäten können bei SOAP Nachrichten durch entsprechende Header unterstützt werden. SOAP Engines, wie z.B. Apache Axis, bieten die Möglichkeiten, zusätzliche Handler-Komponenten zu definieren, die dann diese Header verarbeiten. In zukünftigen Erweiterungen können also weitere Handler in der Axis SOAP Engine eingesetzt werden, mit denen die modulare Nutzung von zusätzlichen WS-Spezifikationen möglich ist.

5.3.5 Enterprise Service Bus als Verbindungsglied

In einem Unternehmen werden oft verschiedene Softwarelösungen eingesetzt. Für die Planung von Projekten ein Projektplanungs-Tool, für die Planung von Ressourcen eine ERP-Software, für die Planung einer Produktion eine Produktplanungssoftware, für die Abfrage und die Speicherung von Daten ein Datenbanksystem oder für die Konstruktion von Modellen ein CAD-Programm. Jetzt kommt zusätzlich für die Abfrage von Kontextinformationen die Nexus Plattform hinzu. Das Problem besteht nun darin, dass die Systeme alle eine unterschiedliche Schnittstelle anbieten, eventuell auf unterschiedlichen Plattformen laufen und unterschiedliche Protokolle für die Kommunikation verwenden. Jedes System braucht für den Zugriff auf ein anderes System einen speziellen Adapter. Das macht bei n verschiedenen Systemen insgesamt $n * (n-1)$ Adapter. Zusätzlich kann es vorkommen, dass ein System zeitweise nicht zur Verfügung steht, z.B. weil der Administrator es heruntergefahren hat oder ein Netzwerkproblem besteht. Nachrichten, die an das System gesendet wurden, gehen eventuell verloren.
Eine bessere Lösung wäre, die verschiedenen Systeme über einen gemeinsamen Bus zu verbinden,

einen Enterprise Service Bus (kurz ESB). Der ESB ermöglicht das Einbinden von Systemen in den Bus, kümmert sich um das Weiterleiten von Nachrichten unter den Systemen, ermöglicht die Unterstützung verschiedener Transportprotokolle und sorgt für eine robuste und sichere Kommunikation. Nachrichten gehen nicht verloren und jedes System kennt zur Kommunikation mit den anderen Systemen lediglich eine Schnittstelle, nämlich die Schnittstelle zum ESB.
In Abbildung 5.1 sehen wir exemplarisch die Integration von Nexus in einen ESB, der auf dem Java Business Integration Standard basiert (kurz JBI). Die Verknüpfung der Nexus Plattform und weiterer Systeme, z.B. einer ERP-Software, mit dem ESB könnte zum einen über sog. Verbindungskomponenten erfolgen, im Bild mit BC (Binding Component) bezeichnet. Für die Integration der einzelnen Nexusdienste müssten diese Verbindungskomponenten entworfen werden. Zum anderen könnte die Integration der Nexus Plattform durch die Überführung der hier definierten Integrationsprozesse in die BPEL Engine des ESB erfolgen.

5.4 Richtlinien für die Erstellung von Prozessen

Für die Erstellung von Workflowanwendungen, die über die Integrationsprozesse (siehe Abschnitt 4.3) auf Kontextinformationen der Augmented World zugreifen, sollen hier zusätzlich Richtlinien aufgeführt werden, die den Einstieg in die Modellierung von Workflowanwendungen erleichtern:

- **Entwicklungsumgebung ausnützen**: Eine Entwicklungsumgebung unterstützt den Entwickler bei der Programmierung durch nützliche Funktionen. Der *Oracle JDeveloper* ist als Entwicklungsumgebung bereits in der Distribution der Workflowmaschine *Oracle BPEL Server 10.1.2* integriert. Der JDeveloper bietet u.a. einen BPEL Editor für die grafische Erstellung von BPEL Prozessbeschreibungen. Zusätzlich bietet die Entwicklungsumgebung die Möglichkeit, sich zur BPEL Engine von Oracle verbinden zu lassen und den BPEL Prozess auf dieser zu installieren. Weitere nützliche Funktionen sind beispielsweise ein Modul zur Erstellung von XPath Ausdrücken oder ein Modul zur Erstellung von XSLT Stylesheets. Die Entwicklungsumgebung bietet dabei jederzeit die Möglichkeit, von der grafischen Darstellung in die Codedarstellung zu wechseln und erlaubt somit auch den direkten Eingriff in den Programmcode. Diese Funktionen vermindern durch die integrierte Syntaxprüfung die Fehlerquote im Code und erlauben eine schnellere Entwicklung.
- **Synchrone oder asynchrone Verarbeitung**: Zu Beginn der Entwicklung eines Prozesses sollte geklärt werden, ob der Prozess als synchroner oder asynchroner Dienst zur Verfügung gestellt werden soll. Zur Entscheidungsfindung können folgende Fragen herangezogen werden: Soll der Dienst den übergeordneten Prozessschritt blockieren? Wird vom aufrufenden Prozess sofort eine Antwort erwartet? Handelt es sich beim Prozess um einen Vorgang, der Ergebnisse nach einer absehbaren Zeit liefert?
- **Prozessablauf und Prozesshierarchie skizzieren**: Bei der Beispielanwendung aus Abschnitt 4.6 ist zu erkennen, dass der ParcelService Prozess in Teilprozesse aufgeteilt wurde. Dies verbessert nicht nur die Lesbarkeit des Prozessmodells und schafft ein besseres Verständnis für den Ablauf, sondern es fördert auch die Wiederverwendbarkeit von Prozessteilen in anderen Prozessmodellen. Für diesen Zweck können z.B. BPMN-Diagramme verwendet werden [BPMN].
- **Integrationsprozess auswählen**: Während der Modellierung des Prozesses ist abzuklären, welche Funktionalität der Nexus Plattform benötigt wird, z.B. Abfrage oder Änderung

von Daten. Daraufhin sollte ein geeigneter Integrationsprozess ausgewählt werden. Hier wird auch auf Abschnitt 5.3 verwiesen, in dem die anwendungsspezifische Modellierung zusätzlicher Integrationsprozesse vorgeschlagen wird.

- **Rollenverteilung**: Der Aufruf anderer Prozesse bzw. Web Services erfolgt in BPEL Prozessen über partnerLink Elemente. Bei einem synchronen Aufruf wird nur dem aufgerufenen Prozess die Rolle des Dienstanbieters aus dem partnerLink zugewiesen. Bei asynchronen Aufrufen sollte zusätzlich darauf geachtet werden, dass dem aufrufenden Prozess die Rolle des Dienstempfängers zugeteilt wird.
- **Schnittstelle für eigenen Prozess klären**: Bei der Modellierung eines Prozesses muss vorher geklärt werden, welche Informationen für eine erfolgreiche Verarbeitung benötigt werden. Eine einfache Schnittstelle ermöglicht eine einfache Verwendung durch aufrufende Prozesse, da wenig Daten erforderlich sind. Dafür ist die angebotene Funktionalität eingeschränkt, siehe z.B. ContextEventOnEnterArea mit dem nur On Enter Area Ereignisse registriert werden können. Eine komplexe Schnittstelle bietet eine erweiterte Funktionalität, verlagert jedoch den Aufwand für die Aufbereitung der Eingabedaten in den aufrufenden Prozess, z.B. durch zusätzliche Abfragen oder Transformationen. Die Modellierung einfacher Schnittstellen bietet sich an, sobald ein Prozess öfter verwendet wird. Für die zu übergebenden Daten ist es zudem sinnvoll einen geeigneten Datentyp zu wählen. Für eine Zeitangabe in Minuten ist der XML Schema Datentyp positiveInteger aussagekräftiger als ein string. Durch die Wahl der korrekten Datentypen wird so die Semantik einer Schnittstelle zusätzlich unterstützt.
- **Datenauswahl in der Auswahlbedingung oder in der Transformation**: Bei der Abfrage von Kontextdaten gibt es grundsätzlich zwei Ansätze. Die erforderlichen Kontextinformationen werden entweder in der Auswahlbedingung (restriction Element des AWQL Schema) ausführlich formuliert, so dass die gewünschten Objekte zurückgeliefert werden oder die Auswahlbedingung wird allgemeiner gehalten, so dass eine erweiterte Auswahlmenge zurückgeliefert wird, aus der in einem nachfolgenden Transformationsschritt die interessanten Daten herausgefiltert werden müssen. Diese Regel bezieht sich insbesondere auf alle Integrationsprozesse, in der das restriction Element als Eingabe erwartet wird. Grundsätzlich ist die ausführliche Formulierung einer Bedingung ratsam, da somit der wesentliche Rechenaufwand auf die Nexus Plattform verlagert wird und die zu übertragende Datenmenge geringer ist. Außerdem ist die Definition von logischen Verarbeitungsanweisungen in einem XSL Stylesheet aufwendiger.
- **Namespaces**: Ein wesentlicher Bestandteil in der Arbeit mit allen Technologien, die auf der XML Spezifikation aufbauen, sind die Namensräume (siehe Abschnitt 2.2.1). Bei der Modellierung von Prozessen sollte deshalb immer auf die Verwendung geeigneter Namensräume geachtet werden. Ein häufiger Fehler bei der Arbeit mit XML Datentypen oder BPEL Prozessen ist die fehlerhafte oder fehlende Verwendung von Namensräumen. Es werden z.B. Datentypen mit gleichem Namen im selben Namensraum definiert, es werden Datentypen in den falschen Namensraum importiert oder ein Element wird über einen falschen Namensraum referenziert. Ein Übersicht über die verwendeten Namensräume in Nexus ist in Tabelle 4.1 gegeben.
- **XSLT Editor und XSLT Debugger**: Bei der Erstellung von XSLT Stylesheets ist es vorteilhaft, einen XSLT Editor und XSLT Debugger zu verwenden. Der XSLT Editor erleichtert die Erstellung von XSLT Stylesheets durch Syntaxhervorhebung oder Bereitstellung von Code Fragmenten. Der XSLT Debugger ermöglicht die Überprüfung von

XSLT Anweisungen anhand von Beispielanfragen.

- **Ausnahmebehandlung**: Bei der Modellierung von Prozessen sollten alle möglichen Szenarios berücksichtigt werden, die außerhalb eines normalen Ablaufs liegen. Diese Ausnahmefälle können entweder durch die Definition von Ausnahmen erfolgen oder durch die Rückgabe von Fehlercodes. Ausnahmen müssen an entsprechender Stelle im Prozess ausgelöst werden. Der aufrufende Prozess sollte diese Ausnahmen dann in einem catch-Block abfangen und entsprechende Aktionen durchführen (siehe Abschnitt 5.3). Falls Fehlercodes zurückgegeben werden, so muss dies dem aufrufenden Prozess bekannt gemacht werden, z.B. in einer nicht-formalen Beschreibung des Prozesses.
- **Variablen für AWML-Daten anlegen**: Für den Austausch von Modelldaten wird das AWML Format verwendet. In Abschnitt 4.6 wurde die Verarbeitung der AWML Daten in einer Beispielanwendung aufgezeigt. Für die Verarbeitung von AWML Daten bietet es sich an, die generischen Typen aus der AWML Nachricht einer Variable im Prozess zuzuordnen, in der jeweils der spezielle Typ des Elements definiert ist. Dies erleichtert den Zugriff auf die Daten, da in späteren Prozessaktivitäten reguläre XPath Anweisungen für die Abfrage der Daten verwendet werden können.
- **Fehleranalyse**: Bei der Ausführung von Prozessen können an vielen Schnittstellen Fehler auftauchen. Viele BPEL Engines bieten für ihre installierten Prozesse Managementfunktionen an, wie auch der Oracle BPEL Server. Im Managementbereich besteht die Möglichkeit, sich den aktuellen Prozesszustand anzeigen zu lassen und es findet sich hier auch meist ein Hinweis auf den Fehler. Weitere Hinweise lassen sich in den Logdateien des BPEL Servers oder der einzelnen Nexusdienste finden. Für die Prüfung der Web Service Schnittstellen der Nexusdienste bietet es sich an, eine separate Anwendung zu verwenden (siehe [soapUI]). Bei der Fehlersuche sollten zusätzlich die Einschränkungen des Systems beachtet werden (siehe Abschnitt 5.2).
- **Beispielprozesse**: Für die Erstellung von Prozessen können bereits vorhandene Prozesse als Beispiele dienen. Dazu kann z.B. der Prozess aus der Beispielanwendung (siehe Abschnitt 4.6) betrachtet werden. Weitere Beispielprozesse werden auch meist mit den BPEL Engines der Hersteller ausgeliefert. Zusätzliche Informationen zu Beispielprozessen finden sich z.B. bei [Oracle].

In diesem Zusammenhang wird zusätzlich für allgemeinere Regeln zum Aufbau einer serviceorientierten Architektur in Unternehmen auf [HVH06] verwiesen. Die dort vorgeschlagenen Regeln, z.B. Trennung von Komponenten nach Fachbereich oder Trennung nach Zusammenhalt und Kopplung, sollen eine qualitativ höherwertige SOA-Architektur fördern. Die gesamte Softwarearchitektur eines Unternehmens wird dort als *Anwendungslandschaft* bezeichnet und es wird eine Referenzarchitektur vorgestellt, mit der die hier vorgestellte Architektur der Integrationsprozesse auf Basis der Nexus Plattform viele Gemeinsamkeiten hat, wie z.B. die Einteilung der Architektur in Prozesskomponenten (Workflowanwendungen und Integrationsprozesse), Funktionskomponenten (Nexusdienste) und Bestandskomponenten (Kontextserver).

6 Fazit

Mit dieser Arbeit wurde ein Konzept realisiert, dass die Integration von Kontextinformationen in Workflows ermöglicht. Workflows haben dadurch die Möglichkeit, Daten, die von ihrer Umgebung bereitgestellt werden, für die Steuerung des eigenen Prozessablaufs zu verwenden. Dabei wurde für die Bereitstellung der Kontextinformationen die Nexus Plattform [Nexus] genutzt. Die Übertragung des Konzepts auf andere Plattformen, die ebenfalls Kontextdaten zur Verfügung stellen, ist möglich, wobei dies immer gegen die Vorteile der Nexus Plattform, wie z.B. die Skalierbarkeit des Umgebungsmodells, abgewägt werden sollte (siehe Kapitel 3). In Abschnitt 4.6 wurde eine Workflowanwendung entworfen, die eine Abholung und Auslieferung von Paketen in einem geografischen Gebiet unterstützt. Der wirksame Nutzen der Integrationsprozesse bei der Modellierung von Workflowanwendungen wurde dort verdeutlicht und zugleich wurde ein Einblick in ein mögliches Szenario gegeben, in dem eine solche Einbindung Sinn macht. Für die Evaluierung der Ergebnisse sind weitere Szenarien entscheidend, die sich in realen Situation abspielen, d.h. in einem realen Produktionsprozess, in dem Menschen, Maschinen und Werkstücke die wesentlichen Rollen bei der Erstellung eines Produkts spielen. Werkstücke müssen mit spezieller Hardware versehen werden, damit sie von der Sensorik der Maschinen erfasst werden können. Menschen müssen mit der nötigen Hard- und Software und den entsprechenden Fähigkeiten, diese einsetzen zu können, ausgestattet werden. Reale Fertigungsprozesse müssen im Bezug auf die informationstechnische Unterstützung hin analysiert werden und es müssen die entsprechenden Prozessteile durch Workflowbeschreibungen modelliert werden (siehe [WLJ+06]). Erst dadurch sind ausgiebigere Untersuchungen der Vor- und Nachteile dieser Arbeit und die Planung und Durchführung von Erweiterungen des Konzepts möglich.

Literaturverzeichnis

[ADB+99] ABOWD, G. D. ; DEY, A. K. ; BROWN, P. J. ; DAVIES, N. ; SMITH, M. ; STEGGLES, P.: Towards a Better Understanding of Context and Context-Awareness. In: *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, Springer-Verlag, 1999. – ISBN 3–540–66550–1, S. 304–307

[AFG+07] ARDISSONO, L. ; FURNARI, R. ; GOY, A. ; PETRONE, G. ; SEGNAN, M.: *A framework for the management of context-aware workflow systems*. 2007

[All01] ALLEN, R.: Workflow: An Introduction. In: *Workflow Handbook 2001* (2001), S. 15–38

[ANGP03] ARRUDA, C. R. E. ; NETO, R. B. ; GRACA PIMENTEL, M. da: Open context-aware storage as a Web Service. In: *Middleware Workshops*, 2003, S. 81–87

[Axis] APACHE SOFTWARE FOUNDATION, AXIS DEVELOPMENT TEAM: *Web Services - Axis*. `http://ws.apache.org/axis/`, Abruf: 8. Juli 2007

[Bau03] BAUER, M.: *Präsentation „Event Service" - Nexus Mitarbeitereinführung*. 2003

[BD04] BALDAUF, M. ; DUSTDAR, S.: A Survey on Context-Aware Systems / Technical University of Vienna. 2004. – Forschungsbericht

[BDG+04] BAUER, M. ; DÜRR, F. ; GEIGER, J. ; GROSSMANN, M. ; HÖNLE, N. ; JOSWIG, J. ; NICKLAS, D. ; SCHWARZ, T.: *Information Management and Exchange in the Nexus Platform Version 2.0*. Stuttgart, 2004

[Ben04] BENGEL, G.: *Grundkurs Verteilte Systeme*. Friedr. Vieweg & Sohn Verlag, 2004. – ISBN 3528257385

[BJRW03] BAUER, M. ; JENDOUBI, L. ; ROTHERMEL, K. ; WESTKÄMPER, E.: Grundlagen ubiquitärer Systeme und deren Anwendung in der „Smart Factory". (2003)

[BPEL] IBM DEVELOPERWORKS: *Business Process Execution Language for Web Services version 1.1*. `http://www-128.ibm.com/developerworks/library/specification/ws-bpel/`, Abruf: 8. Juli 2007

[BPMN] OMG: *BPMN Information Home*. `http://www.bpmn.org/`, Abruf: 8. Juli 2007

[CBL+06] CHEN, S. ; BU, Y. ; LI, J. ; TAO, X. ; LU, J.: Toward Context-Awareness: A Workflow Embedded Middleware. In: *UIC*, 2006, S. 766–775

[CCD+01] CAGLE, K. ; CORNING, M. ; DIAMOND, J. ; DUYNSTEE, T. ; GUDMUNDSSON, O. G. ; MASON, M. ; PINNOCK, J. ; SPENCER, P. ; TANG, J. ; WATT, A.: *Professional XSL*. Wrox Press, 2001. – ISBN 1861003579

[CFJ04a] CHEN, H. ; FININ, T. ; JOSHI, A.: A context broker for building smart meeting rooms. In: *Proceedings of the Knowledge Representation and Ontology for*

Autonomous Systems Symposium, 2004 AAAI Spring Symposium, AAAI Press, 2004, S. 434–441

[CFJ+04b] CHEN, H. ; FININ, T. ; JOSHI, A. ; KAGAL, L. ; PERICH, F. ; CHAKRABORTY, D.: Intelligent Agents Meet the Semantic Web in Smart Spaces. In: *IEEE Internet Computing* 8 (2004), Nr. 6, S. 69–79. http://dx.doi.org/http://dx.doi.org/10.1109/MIC.2004.66. – DOI http://dx.doi.org/10.1109/MIC.2004.66. – ISSN 1089–7801

[CJ02] CHAPPELL, D. A. ; JEWELL, T.: *Java Web Services*. O'Reilly & Associates, 2002. – ISBN 0596002696

[CSS] W3C, CSS WORKING GROUP: *Cascading Style Sheets*. http://www.w3.org/Style/CSS/, Abruf: 8. Juli 2007

[Dav93] DAVENPORT, T. H.: *Process innovation: reengineering work through information technology*. Harvard Business School Press, 1993. – ISBN 0875843662

[DFG] DFG: *DFG - Sonderforschungsbereiche - Detailsicht Sonderforschungsbereich 627*. http://www.dfg.de/forschungsfoerderung/koordinierte_programme/sonderforschungsbereiche/liste/sfb_detail_627.html, Abruf: 8. Juli 2007

[Eng93] ENGLANDER, R.: *Java and SOAP*. O'Reilly & Associates, 1993. – ISBN 0596001754

[EPSG] EPSG: *OGP Surveying & Positioning Committee*. http://www.epsg.org/, Abruf: 8. Juli 2007

[Far07] Gespräche mit Tobias Farrell zum Thema Lokations- und Ereignisdienst. 2007.

[FHV06] FARELL, T. ; HÖNLE, N. ; VOLZ, S.: *Navigation Component Report Version 1.0*. Stuttgart, 2006

[GPZ05] GU, T. ; PUNG, H. K. ; ZHANG, D. Q.: A service-oriented middleware for building context-aware services. In: *Journal of Network and Computer Applications* 28 (2005), Nr. 1, S. 1–18. http://dx.doi.org/http://dx.doi.org/10.1016/j.jnca.2004.06.002. – DOI http://dx.doi.org/10.1016/j.jnca.2004.06.002. – ISSN 1084–8045

[Gro07] Gespräche mit Matthias Großmann zum Thema Kartendienst, Kontextserver und Augmented World. 2007.

[HIR01] HENRICKSEN, K. ; INDULSKA, J. ; RAKOTONIRAINY, A.: Infrastructure for Pervasive Computing: Challenges. In: *GI Jahrestagung (1)*, 2001, S. 214–222

[HVH06] HUMM, B. ; VOSS, M. ; HESS, A.: Regeln für serviceorientierte Architekturen hoher Qualität. In: *Informatik Spektrum* 29 (2006), Nr. 6, S. 395–411

[HZ03] HEIN, M. ; ZELLER, H.: *Java Web Services*. Addison-Wesley, 2003. – ISBN 3827320712

[InfoSet] W3C, INFOSET WORKING GROUP: *XML Information Set (Second Edition)*. http://www.w3.org/TR/xml-infoset/, Abruf: 8. Juli 2007

[JBS97] JABLONSKI, S. ; BÖHM, M. ; SCHULZE, W.: *Workflow-Management: Entwicklung von Anwendungen und Systemen; Facetten einer neuen Technologie*. dpunkt Verlag für digitale Technologie GmbH, 1997. – ISBN 392099373X

[JMG02] J.BECKER ; MUEHLEN, M. zur ; GILLE, M.: Workflow Application Architectures: Classification and Characteristics of Workflow-based Information Systems. (2002), S. 39–50

[JNR05] JARDIM, C. H. ; NETO, R. B. ; RIBAS, H. B.: Web Services Enabling Context-Aware Applications: Lessons Learned by Integrating E-Learning Applications. In: *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, IEEE Computer Society, 2005. – ISBN 0–7695–2452–4, S. 400

[KK04] KEIDL, M. ; KEMPER, A.: Towards context-aware adaptable web services. In: *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, ACM Press, 2004. – ISBN 1–58113–912–8, S. 55–65

[Krü06] KRÜGER, G.: *Handbuch der Java-Programmierung*. Addison-Wesley, 2006. – ISBN 3827323614

[Lia02] LIANG, S.: *Konzeption und Implementierung der Föderationskomponente in Nexus*. Diplomarbeit Nr. 1964. Universität Stuttgart. 2002

[LR00] LEYMANN, F. ; ROLLER, D.: *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000. – ISBN 0130217530

[LR02] LEONHARDI, A. ; ROTHERMEL, K.: Architecture of a Large-Scale Location Service. In: *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, IEEE Computer Society, 2002. – ISBN 0–7695–1585–1, S. 465

[Mah06] MAHAJAN, R.: SOA and the Enterprise – Lessons from the City. In: *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, IEEE Computer Society, 2006. – ISBN 0–7695–2669–1, S. 939–944

[McL01] MCLAUGHLIN, B.: *Java and XML, Second Edition*. O'Reilly & Associates, 2001. – ISBN 0596001975

[Mel07] MELZER, I.: *Service-orientierte Architekturen mit Web Services; Konzepte - Standards - Praxis*. Spektrum Akademischer Verlag, 2007. – ISBN 9783827418852

[Meß01] MESSMER, J.: *Modellierung der Augmented World in Nexus*. Diplomarbeit Nr. 1870. Universität Stuttgart. 2001

[Nexus] UNIVERSITÄT STUTTGART, SFB 627: *SFB 627 Nexus*. `http://www.nexus.uni-stuttgart.de/index.html`, Abruf: 8. Juli 2007

[NexusSchema] UNIVERSITÄT STUTTGART, INSTITUT FÜR INFORMATIK - ANWENDUNGSSYSTEME: *Nexus Schema 2.0.2*. `http://as.informatik.uni-stuttgart.de/as/nexus/schema/2.0.2/`, Abruf: 8. Juli 2007

[Nic03] NICKLAS, D.: *Präsentation „Modellierung in Nexus" - Nexus Mitarbeitereinführung*. 2003

[NJCGGP04] NETO, R. B. ; JARDIM, C. ; CAMACHO-GUERRERO, J. ; GRACA PIMENTEL, M. da: A Web Service Approach for Providing Context Information to CSCW Applications. In: *LA-WEBMEDIA '04: Proceedings of the WebMedia & LA-*

Web 2004 Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress, IEEE Computer Society, 2004. – ISBN 0–7695–2237–8, S. 46–53

[OASIS] OASIS: *OASIS Standards and Other Approved Work*. http://www.oasis-open.org/specs/index.php, Abruf: 8. Juli 2007

[OGC] OGC: *OpenGIS Specifications (Standards)*. http://www.opengeospatial.org/standards, Abruf: 8. Juli 2007

[Oracle] ORACLE CORPORATION: *Oracle BPEL Process Manager*. www.oracle.com/technology/bpel, Abruf: 8. Juli 2007

[OSGi] OSGI ALLIANCE: *OSGi Technology*. http://www.osgi.org/osgi_technology/?section=2, Abruf: 8. Juli 2007

[Pervasive] HOFFNAGLE, G. F.: *Pervasive Computing Preface*. http://www.research.ibm.com/journal/sj/384/preface.html, Abruf: 8. Juli 2007

[SC06] SINGH, A. ; CONWAY, M.: Survey of Context aware Frameworks Analysis and Criticism / Web Service Modeling Framework Initiative. 2006. – Forschungsbericht

[Sch03] SCHWARZ, T.: *Präsentation „Nexus Föderation" - Nexus Mitarbeitereinführung*. 2003

[Sch05] SCHOCH, T.: Middleware für Ubiquitous-Computing-Anwendungen. In: *Das Internet der Dinge* B (2005), S. 119–140

[SDA99] SALBER, D. ; DEY, A. K. ; ABOWD, G. D.: The context toolkit: aiding the development of context-enabled applications. In: *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, 1999. – ISBN 0–201–48559–1, S. 434–441

[SOAP] W3C, SOAP WORKING GROUP: *SOAP Specifications*. http://www.w3.org/TR/soap/, Abruf: 8. Juli 2007

[soapUI] EVIWARE SOFTWARE AB: *soapUI*. http://www.soapui.org/, Abruf: 8. Juli 2007

[SVG] W3C, SVG WORKING GROUP: *Scalable Vector Grapics (SVG)*. http://www.w3.org/Graphics/SVG/, Abruf: 8. Juli 2007

[TAB01] TRUONG, K. N. ; ABOWD, G. D. ; BROTHERTON, J. A.: Who, What, When, Where, How: Design Issues of Capture & Access Applications. In: *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, Springer-Verlag, 2001. – ISBN 3–540–42614–0, S. 209–224

[Ubiquitous] WEISER, M.: *The Computer for the 21st Century*. http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html, Abruf: 8. Juli 2007

[UML] OMG INC.: *Object Management Group - UML*. http://www.uml.org/, Abruf: 8. Juli 2007

[WBFT04] WANG, D. ; BAYER, T. ; FROTSCHER, T. ; TEUFEL, M.: *Java Web Services mit Apache Axis*. Software und Support Verlag, 2004. – ISBN 3935042574

[WCL+05] WEERAWARANA, S. ; CURBERA, F. ; LEYMANN, F. ; STOREY, T. ; FER-

GUSON, D.: *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005. – ISBN 0131488740

[WfMC] WFMC: *WfMC.org Standards - Reference Model*. http://www.wfmc.org/standards/referencemodel.htm, Abruf: 8. Juli 2007

[WKNL07] WIELAND, M. ; KOPP, O. ; NICKLAS, D. ; LEYMANN, F.: Towards Context-Aware Workflows. In: *Proceedings of the Ubiquitous Mobile Information and Collaboration Systems (Caise'07 Workshop), Trondheim, Norway, June 11-12th, 2007*, Springer Verlag, 2007

[WLJ+06] WIELAND, M. ; LEYMANN, F. ; JENDOUBI, L. ; NICKLAS, D. ; DÜRR, F.: Task-orientierte Anwendungen in einer Smart Factory. In: *Mobile Informationssysteme - Potentiale, Hindernisse, Einsatz. Proceedings MMS´06*, Gesellschaft für Informatik, 2006 (Lecture Notes in Informatics (LNI); P-76). – ISBN 3–88579–170–6, S. 139–143

[WSDL] W3C, WSDL WORKING GROUP: *Web Service Definition Language (WSDL)*. http://www.w3.org/TR/wsdl, Abruf: 8. Juli 2007

[WSFL] IBM DEVELOPERWORKS: *WSFL in action, Part 1*. http://www.ibm.com/developerworks/webservices/library/ws-wsfl1/, Abruf: 8. Juli 2007

[Wut06] WUTKE, D.: *Erweiterung einer Workflow-Engine zur Unterstützung von parametrisierten Web Service Flows*. Diplomarbeit Nr. 2401. Universität Stuttgart. 2006

[XLANG] MICROSOFT DEVELOPER NETWORK: *XLANG/s Language*. http://msdn2.microsoft.com/en-us/library/aa577463.aspx, Abruf: 8. Juli 2007

[XML] W3C, XML ACTIVITY: *Extensible Markup Language (XML)*. http://www.w3.org/XML/, Abruf: 8. Juli 2007

[XMLSchema] W3C, XML SCHEMA WORKING GROUP: *W3C XML Schema*. http://www.w3.org/XML/Schema, Abruf: 8. Juli 2007

[XPath] W3C, XPATH WORKING GROUP: *XML Path Language (XPath)*. http://www.w3.org/TR/xpath, Abruf: 8. Juli 2007

[XSLT] W3C, XSLT WORKING GROUP: *XSL Transformations (XSLT)*. http://www.w3.org/TR/xslt, Abruf: 8. Juli 2007

A Datentypen und Datenaustauschformate in Nexus

Für die Modellierung von Integrationsprozessen, die eine Integration von Nexusdiensten in eine Workflowumgebung ermöglichen, ist die Kenntnis der wesentlichen Formate und Datentypen von Nexus fundamental. Für die Modellierung von Daten in der Augmented World werden von der Nexus Plattform Standardklassen und Standardattribute zur Verfügung gestellt. Für den Zugriff auf die Funktionen der Nexus Plattform und die damit zusammenhängende Kommunikation mit den jeweiligen Diensten werden verschiedene Schnittstellenbeschreibungen von der Nexus Plattform zur Verfügung gestellt. Die einzelnen Datentypen und Schnittstellenbeschreibungen werden in den jeweiligen Abschnitten genauer betrachtet. Des Weiteren gibt es zusätzliche Elemente der Nexus Plattform, die für ein Verständnis wichtig sind, wie z.B. geografische Formate oder die eindeutigen Bezeichner von Objekten. Die zusammengetragenen Informationen basieren zum größten Teil auf den Angaben in [BDG+04], den Beschreibungen der Nexus Schemata bei [NexusSchema], aus Gesprächen mit [Gro07] und [Far07], den Angaben in der Nexus-Wiki, sowie der Analyse von Programmcode aus dem Nexus Projekt.

A.1 NSAT, NSAS, NSCS und erweiterte Schemata

Für die Modellierung von Daten in Nexus wird das Augmented World Schema verwendet. Dabei basiert das Schema auf Standarddatentypen, Standardattributen und Standardklassen, die jeweils in XML Schema beschrieben sind. Für die Standarddatentypen gibt es das Nexus Standard Attribute Types Schema (NSAT), für die Standardattribute gibt es das Nexus Standard Attribute Schema (NSAS) und für die Standardklassen gibt es das Nexus Standard Class Schema (NSCS). Die Elemente aus dem NSAT Schema basieren dabei auf grundlegenden Datentypen, die bereits in XML Schema definiert sind (siehe Abschnitt 2.2.1), u.a. String, Integer, Float, Boolean. Aus diesen Datentypen werden einfache und komplexere Datentypen speziell für das NSAT Schema gebildet. Beispiele für einfache Nexusdatentypen sind NexusStringType, NexusBooleanType und NexusIntegerType, die auf den jeweiligen XML Datentypen aufsetzen, aber im Namensraum von NSAT neu definiert wurden. Beispiele für komplexere Datentypen aus NSAT sind NexusPoint-Type und NexusGeometryType. Beide Datentypen sind für die Spezifikation von geografischen Attributen wichtig. Mit NexusGeometryType können komplexere Geometrien wie z.B. Polygone angegeben werden und mit NexusPointType werden geografische Positionen spezifiziert. Beide Datentypen basieren für die Repräsentierung von Geodaten auf einer Auswahl zwischen dem Well-Known Text Format, kurz WKT, und der Geography Markup Language, kurz GML (siehe Abschnitt A.2).
Da ein Objekt in der Augmented World nichts mit einfachen Datentypen der Informationstechnik anfangen kann und stattdessen eher bestimmte Eigenschaften, wie z.B. eine Farbe oder eine Position, hat, müssen die Datentypen aus dem NSAT verwendet werden, um Attribute von Objekten zu beschreiben. Vorgaben für wesentliche Attribute von Objekten der Augmented

Listing A.1: Positionsangabe in WKT und GML

```
<nsat:WKT srscode="4326">
    POINT (48.7436029 9.096517)
</nsat:WKT>

<gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326"
           xmlns:gml="http://www.opengis.net/gml">
    <gml:coordinates>48.7436029,9.096517</gml:coordinates>
</gml:Point>
```

World finden sich deshalb im NSAS wieder. Beispiele für Attribute sind „address" für die Modellierung von Adressen, oder „pos" für die Modellierung von geografischen Positionen. Für alle Attribute aus dem NSAS gilt dabei, dass sie von einem übergeordneten Attribut abstammen, dem *NexusAttribute*. NexusAttribute ist ein abstraktes Element welches nicht in XML Dokumentinstanzen auftauchen kann. Es kann jedoch, ähnlich der Vererbung in der objektorientierten Welt, abgeleitet werden. Um die Integration neuer Daten zu ermöglichen und dennoch eine gemeinsame Plattform für unterschiedliche Anwendungen anzubieten, wurde dieser generische Ansatz verwendet, um jedes Attribut in Nexus verarbeiten zu können. Ein Attribut aus dem NSAS kann ebenso über Metadaten verfügen. Metadaten sind dabei Informationen zu den jeweiligen Daten, die in den Attributen angegeben sind. Sie können hilfreich für die weitere Verarbeitung in einer Anwendung sein. Ein Beispiel für den Einsatz von Metadaten wäre die Spezifizierung der Relevanz von Positionsdaten, angegeben durch die Zeit als Metadatum, zu der die Position zuletzt festgestellt worden ist. Metadaten werden in Nexus Attributen durch das Element *meta* angegeben.
Wenn wir die Eigenschaften von Objekten im NSAS festgelegt haben, so brauchen wir jetzt noch die Objekte, denen wir die Eigenschaften anheften können. Deshalb wurden im NSCS die wesentlichen Klassen von Objekten für die Nexus Plattform definiert. Als Beispiel können die Klassen *Building* oder *MobileObject* angeführt werden. Building steht allgemein für alle Objekte die als Gebäude angesehen werden können. Dabei hat Building z.B. die NSAS-Attribute *address* für die Adresse des Gebäudes oder *constructionDate* für das Jahr der Erbauung. MobileObject ist eine Klasse zur Modellierung von beweglichen Objekten. Bewegliche Objekte haben u.a. die Eigenschaften *speed* für die Geschwindigkeit und *direction* für die Richtung, in die sie sich momentan bewegen. Diese Eigenschaften sind ebenso im NSAS definiert. Ebenso wie beim NSAS, leiten auch die Klassen im NSCS von einer gemeinsamen Basisklasse ab, dem *NexusObject*. Die Gründe sind die gleichen wie bei den Attributen. Die Nexus Plattform liefert die Möglichkeit verschiedene Objekte unterschiedlicher Klassen zu verwalten, da alle Objekte von einem grundlegenden Typ ableiten.
Will man zusätzliche Attribute oder Klassen definieren, um spezielle Objekte und Eigenschaften von Objekten modellieren zu können, so bietet die Nexus Plattform die Möglichkeit dies durch erweiterte Klassen- und Attributschemata zu tun. Diese werden dann als *Extended Class Schema* bzw. als *Extended Attribute Schema* bezeichnet. Die Erstellung der erweiterten Schemata muss dabei auf den Standardschemata NSAT, NSAS und NSCS basieren. Ein Beispiel wäre die Erweiterung der Nexus Plattform um Objekte aus dem industriellen Bereich. Dann kann anstatt der generellen Klasse Building für eine Werkstatt oder eine Gießerei auch die entsprechende Repräsentation erstellt werden. Die neue Klasse *Werkstatt* hätte dann ebenfalls neu definierte Attribute wie z.B. „AnzahlHandwerker" oder „VorhandeneWerkzeuge".

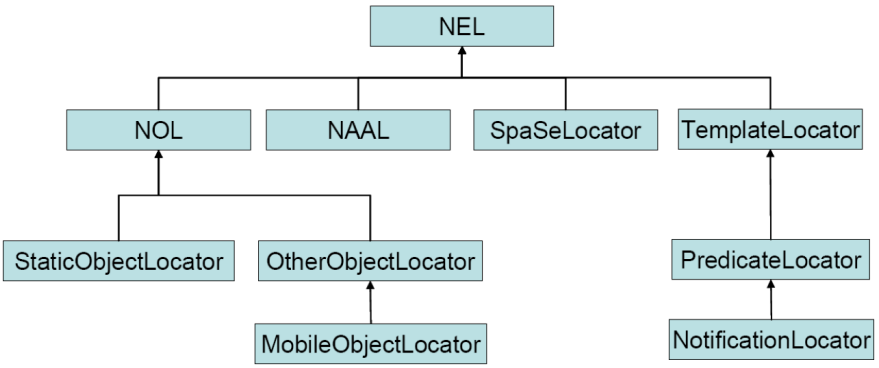

Abbildung A.1: Übersicht Nexus Bezeichner [BDG+04]

A.2 WKT und GML

Die Beschreibung und der Austausch von geografischen Daten sind u.a. für die Angabe von Objektpositionen oder Gebäudeumrissen wichtig. WKT und GML sind zwei standardisierte Formate für die Beschreibung von Geometriedaten, die von Nexus unterstützt werden. Beide Formate sind Standards des Open Geospatial Consortiums (OGC). WKT ist eine Beschreibung von Geodaten im Textformat. Mögliche Objekte die spezifiziert werden können sind z.B. POINT für Positionen, LINESTRING für Strecken oder POLYGON für Flächen. GML basiert auf XML und bietet die Möglichkeit zur Modellierung komplexer Geo-Objekte. GML avanciert aufgrund seiner Mächtigkeit und der Strukturierung durch XML zum Nachfolger von WKT. Beispiele für die Beschreibung von Positionen sehen wir in Listing A.1. Beide Formate benötigen die Angabe eines geografischen Referenzsystems, dem sog. Spatial Reference System (SRS) oder auch Coordinate Reference System (CRS) genannt. Für die exakte Bestimmung von Gebieten und Positionen auf der Erde muss man sich auf ein solches Referenzsystem einigen. Diese Referenzsysteme verwenden für die Bestimmung der Koordinaten auf der Erde unterschiedliche Referenzellipsoide, geometrische Formen die mathematisch beschrieben werden können und u.a. die Krümmung der Erde berücksichtigen. Dabei unterscheidet man globale Referenzsysteme, welche zur Ortsbestimmung für die gesamte Erdkugel gewählt werden können und projizierte Systeme, welche für bestimmte Regionen auf der Erde erstellt werden und eine genauere Positionsbestimmung erlauben. Das Referenzsystem wird durch einen Code oder einen anerkannten Bezeichner spezifiziert. Bisherige Nexus Projekte verwenden dafür die Bezeichner der European Petroleum Survey Group Geodesy (EPSG). Dabei wurden die EPSG-Codes 4326 für das globale System WGS84 (World Geodetic System 1984) verwendet und Code 32632 für ein projiziertes Koordinatensystem, welches auf WGS84 basiert, aber speziell für die Region um Deutschland geeignet ist. Mehr zu den Referenzsystemen findet sich bei [EPSG], zum GML- und WKT-Format bei [OGC].

A.3 Nexus Locators

Zur eindeutigen Identifizierung von Objekten auf der Nexus Plattform wird ein Bezeichnersystem festgelegt. Dieses Bezeichnersystem legt bislang die Form der Bezeichner u.a. für mobile Objekte (MobileObjectLocator), statische Objekte (StaticObjectLocator), Kontextserver (SpaSeLocator),

Listing A.2: AWQL Austauschformat

```
<element name="awql">
 <complexType>
  <sequence>
   <element ref="aa" minOccurs="0" maxOccurs="unbounded"/>
   <element ref="ecs" minOccurs="0" maxOccurs="unbounded"/>
   <element ref="srs" minOccurs="0" maxOccurs="unbounded"/>
   <element ref="geoDataFormat" minOccurs="0" maxOccurs="unbounded"/>
   <element ref="restriction" minOccurs="0"/>
   <element ref="nearestNeighbor" minOccurs="0"/>
   <choice>
    <element ref="include" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="exclude" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="update"/>
    <element ref="append"/>
    <element ref="delete"/>
   </choice>
  </sequence>
 </complexType>
</element>
```

Augmented Areas (Nexus Augmented Area Locator, NAAL) und Ereignisse (TemplateLocator, PredicateLocator und NotificationLocator) fest. Beispiele für die verschiedenen Bezeichner in Nexus (*Nexus Locator*) finden sich in [BDG+04]. Verwendet werden die Nexus Locators z.B. beim Einfügen neuer Objekte in das Augmented World Model (StaticObjectLocator oder MobileObjectLocator), zur gezielten Abfrage von Daten auf bestimmten Kontextservern (NAAL) oder zur Registrierung von bestimmten Ereignisarten (TemplateLocator). In Abbildung A.1 sehen wir eine Übersicht der verschiedenen Bezeichner.

A.4 AWQL, AWML und CRL

Die Kontextserver und die Föderation verwenden zum Austausch von Daten ihre eigenen Formate. Für die Abfrage von Daten dient die Augmented World Query Language (AWQL). Zur Modellierung der Augmented World wird die Augmented World Modeling Language (AWML) benutzt. Für beide Formate existiert eine Beschreibung in XML Schema. In Listing A.2 sehen wir auszugsweise das XML Schema für das AWQL Format. Ein XML Dokument, dass auf diesem Schema basiert, kann im Body einer SOAP Nachricht an das SeifeServlet versendet werden.
Es folgt eine Beschreibung der Elemente des AWQL Formats:

- Mit **aa** kann eine Augmented Area vorgegeben werden. Die Anfrage wird direkt an einen bekannten Kontextserver gerichtet. Als Wert wird ein Bezeichner vom Typ NAAL erwartet (siehe Abschnitt A.3).
- Mit **ecs** können erweiterte Schemata angegeben werden. Dadurch teilt die Anwendung dem Kontextserver mit, welche Objekte sie neben den Standardklassen kennt. Fehlt die Angabe von erweiterten Schemata in ecs, so werden auch keine erweiterten Klassen im Ergebnis zurückgeliefert.
- Mit **srs** kann ein geografisches Referenzsystem (Spatial Reference Systems) angegeben werden. Die Anwendung teilt dem Kontextserver dadurch mit, welche geografischen

Referenzsysteme sie kennt. Entsprechend dem Referenzsystem werden die geografischen Daten der Objekte ausgegeben.

- In **geoDataFormat** wird angegeben, welches Format für die Darstellung der geografischen Daten verwendet werden soll, WKT oder GML.
- **restriction** enthält die Bedingung für die Auswahl der Objekte ähnlich einer Where-Klausel in SQL Anfragen an eine Datenbank (s.u.). Dadurch ist es möglich, bestimmte Objekte auszuwählen (Auswahlmenge).
- Mit **nearestNeighbors** kann zusätzlich die Auswahl der Objekte auf die **n** nächsten Nachbarn beschränkt werden. Dabei können zusätzlich durch eine verschachtelte Bedingung (restriction Element) die für die Auswahl zu betrachtenden Attribute eingegrenzt werden (s.u.).
- Mit **include** können Typen von Objektattributen angegeben werden, die für die Auswahlmenge angezeigt werden sollen. Es werden nur diejenigen Attribute bei den Objekten der Auswahlmenge angezeigt, die auch mit include ausgewählt wurden. Für jeden Attributtyp ist ein eigenes include Element zu spezifizieren.
- **exclude** hat den gegensätzlichen Effekt zum include. Hier werden Attributtypen angegeben, die nicht bei den Objekten der Auswahlmenge erscheinen sollen. Auch hier muss für jeden Attributtyp der nicht angezeigt werden soll ein eigenes exclude Element spezifiziert werden.
- Mit **update** kann ein Attributwert aktualisiert werden. Dabei wird der Attributtyp angegeben und der neue Wert für das entsprechende Attribut. Das Attribut wird bei allen Objekten der Auswahlmenge aktualisiert, sofern es dort vorkommt.
- Mit **append** kann den Objekten der Auswahlmenge ein Attribut hinzugefügt werden. Es wird der Attributtyp angegeben und der Wert des Attributs.
- **delete** löscht die Auswahlmenge. Das delete Element wird ohne zusätzliche Angaben gesetzt.

Die Elemente include, exclude, update, append und delete werden im AWQL Format als Filter bezeichnet. Der Filter wird dabei auf die vom Kontextserver ermittelte Auswahlmenge angewendet, die mit der Bedingung im restriction Element ermittelt wurde. Die Verwendung von Filtern ist optional. Außerdem kann in einer AWQL Anfrage nur ein Filter angegeben werden, weshalb die Filter auch in einem XML Choice Element deklariert sind.
Ein wesentliches Element des AWQL Formats ist das restriction Element. Mit dem restriction Element lässt sich eine Bedingung für die Auswahl von Objekten der Augmented World angeben. Die Bedingung ist dabei eine aussagenlogische Auswertung (Boolean Conjunction) von Vergleichsoperatoren (Comparison Predicates) auf Datentypen. Jeder Vergleichsoperator liefert ein Ergebnis gemäß der zweiwertigen Aussagenlogik, nämlich wahr oder falsch. Folgende Liste zeigt die möglichen Vergleichsoperatoren:

- Für Standarddatentypen, wie z.B. Strings oder Objekttypen, existieren die Operatoren equal, like, greater und less.
- Für geografische Datentypen, wie z.B. Positionen oder Ausmaße von Objekten, gibt es die Operatoren within und intersects.
- Für zeitbasierte Datentypen gibt es die Operatoren temporalBefore, temporalBeforeBegin, temporalAfter, temporalAfterEnd, temporalEqual, temporalEqualBegin, temporalEqualAfter und temporalIntersects.

Listing A.3: AWML Austauschformat

```
<element name="awml">
 <complexType>
  <sequence>
   <element name="nexusobject" minOccurs="0" maxOccurs="unbounded">
    <complexType>
     <sequence>
      <element ref="NexusAttribute" minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
    </complexType>
   </element>
  </sequence>
 </complexType>
</element>
```

Beim Vergleichsoperator wird immer das Attribut angegeben, welches verglichen werden soll (target) und der entsprechende Wert, mit dem verglichen werden soll (referenceValue). Für den Vergleich von Objektnamen kann z.B. der equal Vergleichsoperator verwendet werden, im target wird dabei das Attribut „name" angegeben und im referenceValue der eigentliche Name, den das Objekt haben soll. Eine detaillierte Beschreibung der Bedeutung der unterschiedlichen Vergleichsoperatoren findet sich in [BDG+04]. Die Vergleichsoperatoren können zusätzlich über eine Konjunktion (logisches UND), eine Disjunktion (logisches ODER) oder über eine Negation (logisches Verneinung) verbunden werden. Die Auswertung der Bedingung erfolgt entsprechend den Regeln der Aussagenlogik. Beispiel: Zwei equal Vergleichsoperatoren die über ein UND verknüpft sind bedeuten logischerweise, dass beide Vergleichsoperatoren wahr sein müssen, damit die gesamte Bedingung wahr ist. Es werden somit nur Objekte zurückgegeben, bei denen beide Vergleichsoperatoren den Wahrheitswert wahr besitzen. Eine Bedingung könnte so formuliert werden, dass z.B. nach allen Handwerkern gesucht wird, die sich momentan in der Fertigungshalle befinden und die die Qualifikation zur Reparatur eines Gabelstaplers vorweisen können.
Eine weitere Besonderheit des AWQL Formats zur Auswahl von Objekten ist das nearestNeighbor Element. Mit diesem Element kann die Auswahl auf n benachbarte Objekte eingegrenzt werden. Dabei wird ein geografischer Referenzpunkt angegeben, zu dem die Objekte benachbart sein sollen. Zusätzlich wird das Attribut angegeben, welches für die räumliche Ordnungsfunktion verwendet werden soll. Dies ist üblicherweise das Attribut, welches die Position bei den Objekten angibt (z.B. das pos Attribut aus NSAS). Schließlich wird die Zahl der Objekte angegeben, die gefunden werden sollen.

Die Augmented World Modeling Language, kurz AWML, wird zum Austausch von Modelldaten der Augmented World verwendet. Die verschiedenen Objekte mit ihren unterschiedlichen Eigenschaften werden durch Elemente der AWML ausgedrückt. In Listing A.3 sehen wir das XML Schema für AWML. Das Schema enthält nur ein Element namens *nexusobject*, welches aus dem bereits bekannten *NexusAttribute* des NSAS zusammengesetzt wird. Damit ist das Format eines AWML Dokuments recht einfach gehalten. An diesem Format erkennen wir, wie es bereits beim NSAS und NSCS erwähnt wurde, dass sämtliche Objekte auf der Nexus Plattform vom abstrakten Typ NexusObject sind, und sämtliche Attribute vom abstrakten Typ NexusAttribute. Dies ist ein Schlüsselkonzept der Nexus Plattform, um die Integration neuer Daten zu fördern. Es ist ähnlich der Programmiersprache Java, in der jedes Objekt zusätzlich vom Typ Object

Listing A.4: Element *type* aus NSAS und Typ *Building* aus NSCS

```
<element name="type">
  <complexType>
    <sequence>
      <element name="value" type="NexusTypeType"/>
    </sequence>
  </complexType>
</element>

<complexType name="Building">
  <complexContent>
    <extension base="nscs:BuildingObject">
      <sequence>
        <element ref="nsas:address" minOccurs="0"/>
        <element ref="nsas:contact" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

abgeleitet ist. Dieses Konzept begünstigt neben der einfachen Integration neuer Objektklassen und -attribute auch den Austausch von Modelldaten, da sich diese Generalisierung auch in der AWML wiederfindet. Der Austausch von Modelldaten mit dem AWML Format ist dabei in beide Richtungen möglich. Der Kontextserver sendet Modelldaten seines Ausschnitts der Augmented World im AWML Format an eine anfragende Anwendung. Die Anfrage der Anwendung wurde dabei mit der AWQL formuliert. Zum anderen können neue Modelldaten von einer Anwendung mit AWML spezifiziert werden und an einen Kontextserver gesendet werden. Dieser fügt diese neuen Daten seinem Modell hinzu. Bei der Modellierung muss der Modellierer dafür Sorge tragen, dass nur vorhandene Objektklassen mit ihren jeweils im Schema spezifizierten Attributen verwendet werden (siehe Abschnitt A.1). Ansonsten werden die modellierten Daten nicht dem Augmented World Model hinzugefügt.
Jede Änderung, die von einer Anwendung im Augmented World Model vollzogen wird, wird vom entsprechenden Kontextserver durch die Change Report Language (CRL) bestätigt. Dabei werden, getrennt in Erfolgs- und Misserfolgslisten, alle Objekte aufgeführt, bei denen eine Änderung durchgeführt werden konnte oder nicht. Die Anwendung kann anhand der CRL-Beschreibung erkennen, ob eine Änderung erneut durchgeführt werden soll bzw. ob die Änderungsaktion mit entsprechenden Mittel kompensiert werden muss. Mehr zur CRL findet sich in [BDG+04].
Schließlich soll noch kurz zusammengefasst werden, welche Operationen eine Anwendung mit den jeweiligen Formaten AWQL und AWML durchführen kann. Mit der Augmented World Query Language lassen sich:

1. existierende Objekte mit ihren Attributen aus der Augmented World abfragen (mit include, exclude oder ohne Filter),
2. existierende Attribute von Objekten verändern, indem man den update Filter verwendet (update),
3. neue Attribute an existierende Objekte anhängen, indem man den append Filter verwendet (append) oder
4. existierende Objekte löschen, indem man das delete Element setzt (delete).

Listing A.5: AWQL Anfrage nach Objekten vom Typ Building

```
<awql xmlns:nsas="http://www.nexus.uni-stuttgart.de/1.0/NSAS"
      xmlns:nscs="http://www.nexus.uni-stuttgart.de/1.0/NSCS">
  <restriction>
    <equal>
      <target>nsas:type.nsas:value</target>
      <referenceValue>nscs:Building</referenceValue>
    </equal>
  </restriction>
</awql>
```

Die erste Operation führt im Gegensatz zu den anderen Operationen keine Änderung am Modell durch. Deshalb wird bei 1.) das Ergebnis einer Abfrage auch im AWML Format zurückgegeben. Ergebnisse der anderen Operationen bewirken eine Rückgabe eines Dokuments im CRL Format. Das Kernelement jeder Operation ist dabei die restriction, die die Auswahlmenge der Objekte bestimmt, die an die Anwendung zurückgegeben werden sollen. Die Filter include und exclude erweitern die Funktionalität der Operation bei 1.), indem bestimmte Attribute ein- bzw. ausgeblendet werden können.
Mit AWQL Nachrichten kann eine Anwendung nur Attribute von Objekten einfügen oder ändern bzw. komplette Objekte löschen (s.o.). Mit der Augmented World Modeling Language hingegen kann die Anwendung in das Augmented World Model komplette Objekte samt Attributen einfügen.

A.5 Referenzierung von Schemaelementen

In vielen Anfragedokumenten von Nexus (siehe z.B. AWQL) wird eine Anfrage oft präzisiert, indem sie die Struktur der Attribute und Klassen ausnützt. Es wird dann oft nach speziellen Eigenschaften bzw. speziellen Klassen von Objekten gesucht. Der Programmierer einer Anwendung kennt sich dabei mit der Struktur der Klassen und Attribute in Nexus aus (er kennt zumindest einige Attribute und Klassen aus NSAT, NSAS und NSCS) und kann deshalb gezielte Anfragen erstellen, in denen diese Struktur aufgegriffen wird. In Anfragen referenziert der Programmierer dann die Eigenschaften von Objekten mit der Punktnotation. In Listing A.4 sehen wir die Struktur des Nexus Attributs *type* aus dem NSAS und die Struktur für die Klasse *Building* aus dem NSCS. Folgendes Beispiel: Der Typ bzw. die Klasse eines Objekts in Nexus wird durch das Attribut *type* spezifiziert, der Wert (*value*) für type ist eine Klasse aus den Standardklassen oder einem erweiterten Klassenschema. Mit einer AWQL-Anfrage sollen alle Objekte vom Typ Gebäude zurückgeliefert werden. In Listing A.5 sehen wir, wie in einer AWQL-Anfrage mit der Punktnotation diese Eigenschaft in einer Bedingung (restriction Element, siehe Abschnitt AWQL) referenziert wird. Dabei werden die jeweiligen Namensräume von NSAS und NSCS in die AWQL Anfrage integriert, um die Attribute bzw. Klassen über den Prefix des Namensraumes eindeutig zu kennzeichnen (siehe Abschnitt 2.2.1). Durch die Punktnotation wird das value Element des type Attributs ausgewählt. Als Vergleichswert für die Bedingung *equal* (siehe Abschnitt AWQL) wird die Klasse *Building* aus dem NSCS gewählt. Der Programmierer weiß also, dass der Wert des type Attributs im untergeordneten value Element enthalten ist. Dieser soll dem Wert Building entsprechen.
In Abbildung A.2 ist der Zugriff auf verschiedene Dokumentteile am Beispiel des *address*

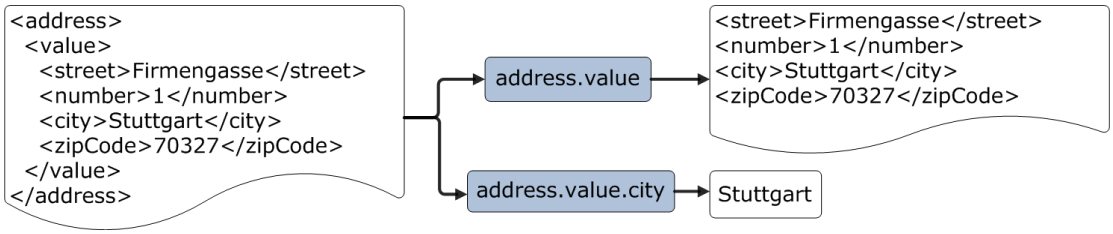

Abbildung A.2: Zugriff auf Dokumentteile mit der Punktnotation

Elements aus dem NSAS abgebildet. Das Element hat ein untergeordnetes Element *value*. Das Element value besteht wiederum aus den Elementen *street*, *number*, *city* und *zipCode*. Der Zugriff auf den Ort geschieht über den folgenden Ausdruck: *address.value.city*. Der Namensraum wurde bei diesem Beispiel ausgelassen. In einer korrekten Anfrage an die Nexus Plattform würde zusätzlich das Element *address* über den Prefix *nsas* an den Namensraum des NSAS gebunden werden. Die Referenzierung des Orts würde dann über den Ausdruck *nsas:address.nsas:value.nsas:city* erfolgen.

A.6 ERL, ENL und EventInfo

Der Ereignisdienst verwendet zur Registrierung von Ereignissen die Event Registration Language (ERL), zur Bestätigung einer Registrierung das EventInfo Format und zur Benachrichtigung über eingetretene Ereignisse die Event Notification Language (ENL). In Listing A.6 sehen wir das Schema für die ERL, deren Elemente hier kurz vorgestellt werden sollen:

- Das Element **templateID** hält einen eindeutigen Bezeichner (siehe Nexus Locators) für die Art des Ereignisses, das mit dem ERL Dokument registriert werden soll. Dieser Locator wird für ein Ereignistyp auf der Nexus Plattform fest vorgegeben und gilt somit für alle Ereignisse von diesem Typ.
- Mit **name** kann ein beliebiger Name für das Ereignis, das man registrieren möchte, angegeben werden. Bei Ereignisrückmeldung ist somit eine benutzerfreundlichere Darstellung gegeben. Der Benutzer kann besser nachvollziehen, um welches Ereignis es sich handelt.
- Ereignisse werden auf den Ereignisquellen beobachtet (siehe Abschnitt 2.1.5). Die Adresse einer bestimmten Ereignisquelle, auf der das Ereignis beobachtet werden soll, kann in **observerURI** explizit angegeben werden.
- Die **predicateId** ist ein eindeutiger Bezeichner (siehe Abschnitt A.3) für das mit dieser ERL spezifizierte Ereignis. Im Vergleich zum Template Locator handelt es sich hier um den Bezeichner für ein definiertes Ereignis eines Ereignistyps. Dieses Element muss nicht gesetzt werden, da es vom Ereignisdienst gesetzt wird.
- Die Ereignisbeobachtung basiert auf wahrscheinlichkeitstheoretischen Berechnungen. In **thresholdProbability** wird der Schwellwert für die Wahrscheinlichkeit, die den Eintritt eines Ereignisses markiert, eingegeben, die vom Aufrufer akzeptiert wird. Ist ein Ereignis mit einer Wahrscheinlichkeit eingetreten, die größer oder gleich dem Schwellwert ist, so wird das Ereignis vom Ereignisdienst gemeldet. Der Schwellwert wird zwischen 0 und 1 angegeben.
- Im Element **parameterList** werden die eigentlichen Parameter für den jeweiligen Er-

Listing A.6: ERL Austauschformat

```
<xs:element name="predicate" type="PredicateType"/>
<xs:complexType name="PredicateType">
 <xs:sequence>
  <xs:element name="templateID" type="NOLType"/>
  <xs:element name="name" type="string" minOccurs="0"/>
  <xs:element name="observerURI" type="anyURI" minOccurs="0"/>
  <xs:element name="predicateId" type="NOLType" minOccurs="0"/>
  <xs:element name="thresholdProbability" type="float"/>
  <xs:element name="parameterList" type="ParameterListType"/>
  <xs:element name="qosParameterList" type="QosParameterListType"/>
  <xs:element name="predicateManagementParameterList" type="PredicateManagementParameterListType"/>
  <xs:element name="predicateEvaluationParameterList" type="PredicateEvaluationParameterListType" minOccurs="0"/>
  <xs:element name="eventNotificationParameterList" type="EventNotificationParameterListType" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
```

eignistyp in Form von Attribut-Werte Paaren angegeben. Die Art bzw. das Vorkommen bestimmter Parameter hängt hier also vom Ereignistyp ab (s.u.).

- **qosParameterList** enthält Attribut-Werte Paare, die Quality of Service Parameter angeben (s.u.).
- **predicateManagementParameterList** enthält Attribut-Werte Paare, mit denen die Funktionalität des Ereignisdiensts eingestellt werden kann (s.u.).
- **predicateEvaluationParameterList** enthält Attribut-Werte Paare, mit denen die Auswertung von Ereignissen gesteuert werden kann (s.u.).
- **eventNotificationParameterList** enthält Attribut-Werte Paare, mit der die Darstellung der Ereignisbenachrichtigung gesteuert werden kann (s.u.).

Verschiedene Ereignistypen können beim Ereignisdienst registriert werden. Davon sind folgende Ereignisse interessant für eine Anwendung:

- *On Enter Area*: Ein mobiles Objekt (object identifier) betritt ein bestimmtes Gebiet (observation area). Das mobile Objekt wird durch einen Bezeichner spezifiziert (siehe Nexus Locators), das Gebiet durch die Angabe der geografischen Repräsentation, beschrieben durch ein geografisches Format (siehe Abschnitt A.2). Beispiel: Ein Arbeiter betritt das Firmengelände - die Aktion der Anwendung darauf könnte das Festhalten des Arbeitsbeginns sein.
- *On Leave Area*: Erhält die gleichen Parameter wie On Enter Area, ist aber das gegensätzliche Ereignis. Beispiel: Ein Arbeiter verlässt das Firmengelände - die Aktion darauf könnte das Festhalten des Arbeitsendes sein.
- *On Meeting*: Zwei mobile Objekte (object identifier und secondary object identifier) treffen sich. Dabei wird zusätzlich die Distanz für ein Treffen angegeben (report distance). Beispiel: Ein Handwerker läuft an einem defekten Gabelstapler vorbei - die Aktion darauf könnte ein Reparaturauftrag an den Handwerker sein.
- *On Register Area*: Ein mobiles Objekt (object identifier) registriert sich in einem bestimmten Gebiet (observation area). Beispiel: Ein Montagearbeiter bekommt einen neuen Arbeitsplatz - Aktion könnte die Zuweisung neuer Aufgaben sein.
- *On Deregister Area*: Das gegensätzliche Ereignis zu On Register Area mit gleichen Parametern. Beispiel: Ein Arbeiter verlässt eine Abteilung - Aktion könnte eine Umstellung der Kostenstelle sein.
- *Continuous Area Update*: Dieses Ereignis meldet die Position aller Objekte (object selector)

in einem bestimmten Gebiet (observation area) nach einem vorgegeben Intervall (report interval). Die Objekte, die an dem Ereignis teilnehmen sollen, werden durch Attribut-Werte Paare spezifiziert, z.B. Fahrzeugart = LKW, Transportgut = Getriebe. Die Spezifikation solcher Attribut-Werte Paare ist beim Ereignisdienst unter dem Begriff *Object Selector* bekannt. Dabei können nur Attribute verwendet werden, die der Ereignisquelle bekannt sind (siehe NSAT, NSAS und NSCS). Beispiel: Die Position von Frachtgütern wird ständig beobachtet - Aktionen könnten die Überwachung und Steuerung des Transports durchführen.

- *Continuous Position Update*: Die Position eines mobilen Objekts (object identifier) wird nach einem vorgegebenen Intervall gemeldet (report interval). Beispiel: Der Fortschritt einer Montage an einem Motor wird überwacht, indem die Position des Motors auf der Fertigungsstraße angezeigt wird.
- *Distance Position Update*: Ein mobiles Objekt (object identifier) hat seit Registrierung des Ereignisses eine vorgegebene Distanz (report distance) überschritten. Dabei ist mit der Distanz der direkte Abstand (Luftlinie) zu dem Punkt gemeint, an dem sich das Objekt bei der Registrierung des Ereignisses befand.

Zusätzlich kann zu den einzelnen erforderlichen Parametern der einzelnen Ereignistypen, die auf der Berechnung von Abständen beruhen (z.B. On Meeting), immer die Genauigkeit für den Abstand angegeben werden (required accuracy), z.B. required accuracy = 10 steht für eine Genauigkeit von 10 Metern.
Für die Bereiche *Quality of Service*, *Evaluation*, *Notification* und *Management* konnten nur wenige Parameter ausgemacht werden. Im Quality of Service Bereich werden Qualitätsansprüche bei der Durchführung der Ereignisbeobachtung definiert, z.B. *maximum clock skew* für die maximale Uhrenabweichung zwischen zwei Systemen oder *maximum delay* für die maximale Zeit zwischen Registrierung und Beobachtung eines Ereignisses. Im Bereich Management können Parameter angegeben werden, die für die Verwaltung des Ereignisdiensts interessant sind, z.B. *registration interval* für die Zeit, nach der ein Ereignis automatisch abgemeldet werden soll. In der Evaluation Liste werden Parameter gesetzt, mit der die Auswertung von Ereignissen gesteuert werden kann, z.B. *blocking interval* für die Zeit, in der das Ereignis nicht erneut gemeldet werden soll, selbst wenn es eintreten sollte. Im Notification Bereich kann Einfluss auf die Darstellung der Benachrichtigung genommen werden, z.B. das Ausblenden bestimmter Rückgabewerte. Die Implementierung von Parametern in diesen Bereichen ist nicht vollständig. Lediglich für *blocking interval* aus dem Evaluation Bereich und für *registration interval* aus dem Management Bereich konnten während der Bearbeitungszeit Implementierungen beim Ereignisdienst festgestellt werden.
Für die Registrierung eines Ereignisses sind folgende Parameter erforderlich: Template Locator, ereignisspezifische Parameter (siehe Ereignisliste), Probability Threshold und das Registration Interval (aus dem Management Bereich). Alle übrigen Parameter sind optional.
Ein Dokument vom Schema EventInfo (Event Information) wird nach der erfolgreichen Registrierung des Ereignisses zurückgegeben. Darin enthalten sind der Predicate Locator, mit dem das Ereignis eindeutig identifiziert werden kann, der Template Locator, mit dem der Typ des Ereignisses ermittelt werden kann, der benutzerfreundliche Name des Ereignisses, der in der Registrierung mit angegeben wurde und die Zeit für die Registrierung und Deregistrierung, die jeweils den Wert der Methode getTime() der Klasse java.util.Date eines Datums enthält.

Die Benachrichtigung über ein Ereignis erfolgt mit der Event Notification Language, kurz ENL. Eine Anwendung kann bei Erhalt eines EventInfo Dokuments den Predicate Locator des

Listing A.7: ENL Austauschformat

```
<xsd:element name="notification" type="NotificationType"/>

<xsd:complexType name="NotificationType">
  <xsd:sequence>
    <xsd:element name="id" type="NOL"/>
    <xsd:element name="predicateId" type="NOL"/>
    <xsd:element name="templateId" type="NOL"/>
    <xsd:element name="name" type="string"/>
    <xsd:element name="service" type="string"/>
    <xsd:element name="server" type="string"/>
    <xsd:element name="counter" type="unsignedLong"/>
    <xsd:element name="scope" type="dateTime"/>
    <xsd:element name="timestamp" type="dateTime"/>
    <xsd:element name="comment" type="string" minOccurs="0"/>
    <xsd:element name="variableList" type="VariableListType"/>
  </xsd:sequence>
</xsd:complexType>
```

registrierten Ereignisses speichern und bei Erhalt einer Benachrichtigung im ENL Format den dort vorhandenen Predicate Locator (siehe Abschnitt A.3) mit dem abgespeicherten Bezeichner vergleichen. Somit kann eine Zuordnung eines Ereignisses in der Anwendung erfolgen. Ein ENL Dokument wird nur versendet, wenn das Ereignis auch eingetreten ist. Interessant sind die Daten eines ENL Dokuments für die Anwendung besonders bei den Ereignissen, bei denen sich die ereignisspezifischen Parameter ändern, z.B. die ständige Positionsänderung bei Continuous Position Update (siehe Liste Ereignistypen). Bei Ereignissen, bei denen sich die Parameter nicht ändern, z.B. On Enter Area, ist eine Auswertung des ENL Dokuments nicht unbedingt notwendig. Falls eine Auswertung erfolgen soll, so muss das variableList Element aus dem ENL Schema verarbeitet werden (siehe Listing A.7). Dieses Element enthält eine Liste von Variablen, die vom Ereignisdienst zu dem besonderen Ereignis mitgesendet werden, z.B. die entdeckten Objekte mit ihren Positionen. Eine Variable ist dabei ein 4-Tupel, welches aus den Elementen name, type, restriction und value besteht. Beispiel: Registriert wurde das Ereignis On Enter Area auf ein bestimmtes Objekt das ein Gebiet betreten wird. In der Liste der Variablen werden beim Eintreten des Ereignisses das Gebiet und das Objekt nochmals aufgeführt. Die Werte der einzelnen Elemente für die Variable Objekt könnten sein:

- name: Entering Object
- type: entsprechende Java Klasse (z.B. Klasse OInfo)
- restriction: Mobile Object
- value: die NOL des Objekts (siehe Nexus Locators)

Von den übrigen Elementen der ENL steht *predicateId* für den Predicate Locator, *templateId* für den Template Locator, *name* für den benutzerfreundlichen Namen des Ereignisses, *service* für den Dienst, der das Ereignis beobachtet hat, *server* für die Adressinformationen des Diensts, *counter* für die Zahl der Beobachtungen des gleichen Ereignisses, *timestamp* für die Zeit, zu der beobachtet wurde, scope gibt die Zeit an bis zu der die Beobachtung gültig ist und *comment* kann vom Ereignisdienst für einen beliebigen Kommentar genutzt werden. Das Element *id* ist eine Erweiterung des bekannten Predicate Locator. Da das gleiche Ereignis mehrmals beobachtet werden kann bzw. soll (Continuous Pos Update), werden diese Ereignisse zusätzlich durch einen

Listing A.8: MapPL Austauschformat

```
<element name="mappl">
  <complexType>
    <sequence>
      <element name="imageSize"/>
      <element name="imageEncoding"/>
      <element name="clippingRectangle"/>
      <element name="drawingRule" minOccurs="0" maxOccurs="unbounded"/>
      <element name="css" minOccurs="0"/>
      <any namespace="http://www.w3.org/2000/svg" minOccurs="0"/>
      <element ref="awql:awql"/>
    <sequence>
  <complexType>
</element>
```

Bezeichner voneinander unterschieden. Weitere Informationen zum ERL, ENL und EventInfo Format finden sich bei [BDG+04].
Vom Ereignisdienst werden Operationen zur Registrierung und Deregistrierung von sämtlichen Ereignissen aus der Ereignisliste (s.o.), zum Initialisieren der Ereignisbeobachtung und zum An- und Abmelden beim Benachrichtigungsdienst (Notification Service) für den Empfang von Benachrichtigungen zur Verfügung gestellt.

A.7 MapPL und MapML

Der Kartendienst (Map Service) von Nexus stellt Objekte der Augmented World auf einer zweidimensionalen geografischen Karte dar. Der Map Service wird über das SeifeServlet angesprochen. Im Falle vom Map Service werden für Anfragen Dokumente vom Format MapPL (Map Predicate Language) gesendet, der Map Service antwortet darauf mit Rückgaben vom Format MapML (Map Modeling Language). Grundsätzlich arbeitet der Kartendienst in zwei Schritten:

- Bestimmung der Objekte, die auf der Karte dargestellt werden sollen. Die geschieht mit einer AWQL Anfrage an die Föderation.
- Darstellung der ausgewählten Objekte auf der Karte mit den vorgegebenen Parametern (s.u.).

In Listing A.8 sehen wir die Elemente der MapPL. Dabei steht das awql Element für ein Dokument im AWQL Format (siehe Abschnitt A.4), und dient somit für die Bestimmung der Objekte, die dargestellt werden sollen. Eine AWQL Anfrage ist also in eine MapPL Anfrage integriert. Alle übrigen Elemente der MapPL dienen der Darstellung der ausgewählten Objekte und sollen hier vorgestellt werden:

- Mit *imageSize* wird die Auflösung der Karte in Pixeln angegeben. Hierfür werden die Attribute Breite (width) und Höhe (height) des Elements verwendet.
- Mit *imageEncoding* wird das Bildformat (format) und die Farbtiefe (bits) angegeben, in dem die Rückgabe erfolgen soll. Hier sind bislang die Werte *svg* für Grafiken im Vektorformat und *jpeg*, *png* bzw. *tiff* für Grafiken im Binärformat möglich. Die Werte werden ebenfalls in den jeweiligen Attributen des Elements gesetzt.

Listing A.9: Element *drawingRule* und entsprechende CSS Deklaration

```
<drawingRule>
  <nexusType>nscs:Building</nexusType>
  <geoElement styleClass="BuildingExtent">
    <awql:target>nsas:extent.nsas:value</awql:target>
    <textElement styleClass="BuildingName">
      <awql:target>nsas:name.nsas:value</awql:target>
    </textElement>
  </geoElement>
</drawingRule>

<css>
  path.CPolygon-BuildingExtent {fill: orange}
  text.BuildingName {font-size: 24}
</css>
```

- Mit *clippingRectangle* wird der geografische Bildausschnitt angegeben, der auf der Karte angezeigt werden soll. Der Ursprung dieses Elements liegt darin, dass mit einer AWQL Anfrage Objekte mit einer großen Ausdehnung abgefragt werden, z.B. Straßen oder langgestreckte Gebäude. Deshalb ist meist der geografische Bereich in der AWQL-Anfrage größer gewählt, damit diese Objekte ebenfalls in der Auswahlmenge erscheinen. Danach wird der Kartenausschnitt, der dargestellt werden soll, mit clippingRectangle ausgewählt, was zur Folge hat, dass die Objekte in der Karte letztendlich vorhanden sind, jedoch Teile der Objekte an den entsprechenden Schnittstellen abgetrennt sind.
- Mit den *drawingRule* Elementen können Regeln für das Zeichnen jedes einzelnen Objekts spezifiziert werden. Die drawingRules hängen vom css Element und vom any Element ab (s.u.).
- Das css Element nimmt Cascading Stylesheet Beschreibungen entgegen. Der Map Service arbeitet intern mit einer Repräsentierung der Karteninformation im SVG Format. Dokumente im SVG Format können für ihre Objekte Formatierungen im CSS Format enthalten. Deshalb werden die hier definierten CSS-Beschreibungen in die SVG-Beschreibung vom Kartendienst übernommen.
- Obwohl der any Teil der MapPL jegliche Beschreibungen aufnehmen kann, werden hier Beschreibungen von Elementen im Scalable Vector Grafik (SVG) Format erwartet. Der Map Service arbeitet grundsätzlich bei der Erstellung der Karteninformation auf einer SVG-Repräsentierung der Daten. Nachdem die Kartendaten vom Map Service im SVG Format generiert worden sind und bevor die Rückgabe an die Anwendung vollzogen wird, wird der Inhalt des any Elements einfach in die SVG-Beschreibung kopiert. Erst danach werden die Karteninformationen zusammen mit der kopierten Ergänzung wahlweise in das Binärformat umgewandelt oder direkt im SVG Format an die Anwendung zurückgeliefert, entsprechend dem Wert im Element imageEncoding. Die Verwendung des any Elements ist z.B. dann nützlich, wenn man die Formatierung eines Objekts komplett im SVG Format vornehmen möchte, z.B. die Darstellung eines VIT als Litfaßsäule.

Detaillierte Informationen zum SVG Format gibt es unter [SVG] und zum CSS Format unter [CSS]. Es soll hier lediglich der Bezug der beiden Formate zum drawingRule Element deutlich gemacht werden. Das drawingRule Element ermöglicht es die Darstellung für Objekte in Nexus zu steuern. Eine Zeichenregel (drawing rule) adressiert durch das Element *nexusType* den Typ

Listing A.10: Auszug einer SVG-Beschreibung

```
<svg>
  <style>
    <!--hier werden die CSS Deklarationen von MAPPL eingesetzt-->
    path.CPolygon-BuildingExtent {fill: orange}
    text.BuildingName {font-size: 24}
  </style>
  <defs>
    <!--hier wird der any Teil von MAPPL eingesetzt-->
  </defs>
  <!-- Es folgen die Objektbeschreibungen -->
  <g>
    <g id="(...StaticObjecLocator...)">
      <path class="CPolygon-BuildingExtent" d="(...Linienzug...)"/>
        <text x="970.1734697519473" y="558.5613342617929">
          <tspan class="BuildingName">Lagerhalle XY</tspan>
        </text>
    </g>
    <!-- Weitere Objektbeschreibungen -->
    <g ...> ... </g>
  </g>
  <!-- Weitere Objektbeschreibungen -->
  <g ...> ... </g>
</svg>
```

des Objekts, für das diese Regel gelten soll (siehe Abschnitt A.1 für Objekttypen). Für dieses Objekt können nun ein oder mehrere *geoElemente* definiert werden, mit denen die geometrischen Attribute eines Objekts formatiert werden sollen (z.B. die Ausmaße eines Raumes oder die Stelle an der sich die Tür im Raum befindet). Dabei werden die Objektattribute, die formatiert werden sollen, in der bereits bekannten Punktnotation referenziert (siehe Abschnitt A.5). Mit *textElement* kann den Objektattributen zusätzlich formatierter Text angeheftet werden. Listing A.9 zeigt ein Beispiel für eine solche Zeichenregel (drawing rule). Hier sehen wir auch den Bezug zur CSS-Deklaration aus dem css Element von MapPL. Die Beziehung der CSS-Deklarationen zu den entsprechenden Geometrieattributen in der Zeichenregel wird durch das Attribut *styleClass* hergestellt. Die CSS-Definition ist hierbei etwas trickreich, da der Programmierer genau wissen muss, welche Geometrie ein Objekt besitzt, denn genau diese Geometrie wird der Map Service später dem frei definierten Namen aus dem Attribut styleClass durch einen Bindestrich in seiner SVG-Beschreibung voranstellen. Da die CSS-Deklaration aus dem css Element vom Map Service einfach direkt übernommen wird, muss der Programmier bereits hier der styleClass die Geometrie voranstellen. Der Grund dafür ist, dass für verschiedene Geometrien die gleiche styleClass Bezeichnung in den geoElement Definitionen verwendet werden kann. Die Darstellung auf der Karte kann jedoch durch Voranstellen der Geometrie bei der entsprechenden styleClass in der CSS-Deklaration unterschiedlich ausfallen.
In Listing A.10 sehen wir einen Auszug aus einer beispielhaften SVG-Beschreibung, die vom Map Service generiert worden ist. Die Bereiche, in die der any-Bereich und das css Element der MapPL Anfrage eingesetzt werden, sind durch Kommentare gekennzeichnet. Das Element *g* steht in der SVG Spezifikation für eine zusammengehörige Gruppe von Geometriedaten (siehe [SVG]). Das Element *path* steht für einen geometrischen Linienzug (Verbindung mehrerer Punkte) und ist von der Klasse CPolygon-BuildingExtent. Das Element *text* enthält den zugehörigen Text der Klasse BuildingName. Für beide Klassen wurde ein Stil in der CSS-Deklaration festgelegt und

Listing A.11: MapML Austauschformat

```
<element name="mapml">
 <complexType>
  <choice>
   <any namespace="http://www.w3.org/2000/svg"/>
   <element name="map"/>
  </choice>
 </complexType>
</element>
```

vom Map Service im Element *style* übernommen. Hier sehen wir auch, dass das Element *defs* den any-Bereich aus der MapPL Anfrage aufnehmen wird. Anwendung findet der any-Bereich z.B. in der Definition von Symbolen im SVG Format, z.B. ein Zylinder als Symbol für einen Virtual Information Tower (VIT). Symbole können für Objekte der Augmented World, deren Geometrie als Punkt spezifiziert ist, über das Attribut *icon* in einem geoElement angegeben werden. Der Wert für icon ist eine URI, an der sich die Symbolbeschreibung befindet. Hierbei ist z.B. „#vit" eine gültige URI auf eine Symbolbeschreibung namens „vit", falls das Symbol mit dem Bezeichner „vit" im any-Bereich der MapPL Anfrage deklariert wird.
Der Kartendienst liefert Ergebnisse auf MapPL Anfragen in der Map Modeling Language (MapML). Dabei ist die Rückgabe entweder vom XML Schema Typ *any*, falls eine Vektorgrafik in SVG-Beschreibung angefordert wurde oder vom XML Schema Typ *base64Binary*, falls Binärdaten in Form von JPEG, PNG oder TIFF Grafiken angefordert wurden. Das Schema ist in Listing A.11 aufgeführt. Weitere Informationen zu MapPL und MapML finden sich in [BDG+04]. Der Map Service bietet Operationen zur Generierung von Karten als Vektor- oder Rastergrafik. Bislang werden die für Rastergrafiken die Formate JPEG, PNG und TIFF unterstützt. Vektorgrafiken werden im SVG Format erstellt. Die Darstellung der Karten kann durch eine Vielzahl von Parameter angepasst werden.

B WSDL Beschreibungen

In diesem Anhang befinden sich die erstellten Schnittstellenbeschreibungen für den Zugriff eines Workflowsystems auf die Nexusdienste Föderation, Ereignis- und Kartendienst. Zusätzlich werden exemplarisch die erstellten WSDL Beschreibungen für die *Integrationsprozesse ContextQuery* und *ContextQueryExclude* aufgeführt. Auf diese Beschreibungen wird z.B. im Feinentwurf Abschnitt 4.3 und in der Implementierung Abschnitt 4.4 verwiesen.

Listing B.1: MapService.wsdl

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions name="webmapservice"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:tns="http://www.nexus.uni-stuttgart.de/1.0/webmapservice/"
                  xmlns:mapplns="http://www.nexus.uni-stuttgart.de/2.0/MapPL"
                  xmlns:mapmlns="http://www.nexus.uni-stuttgart.de/2.0/MapML"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webmapservice/"
                  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webmapservice/"
                xmlns:mapplns="http://www.nexus.uni-stuttgart.de/2.0/MapPL">
      <xsd:import namespace="http://www.nexus.uni-stuttgart.de/2.0/MapPL"
        schemaLocation="http://w3studi.informatik.uni-stuttgart.de/~kaczmapr/nexus/NexusMapPLSchema.xsd"/>
    </xsd:schema>
    <xsd:schema targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webmapservice/"
                xmlns:mapmlns="http://www.nexus.uni-stuttgart.de/2.0/MapML">
      <xsd:import namespace="http://www.nexus.uni-stuttgart.de/2.0/MapML"
        schemaLocation="http://w3studi.informatik.uni-stuttgart.de/~kaczmapr/nexus/NexusMapMLSchema.xsd"/>
    </xsd:schema>
    </wsdl:types>
    <wsdl:message name="mapplDocument"><wsdl:part name="parameters" element="mapplns:mappl"/></wsdl:message>
    <wsdl:message name="mapmlDocument"><wsdl:part name="parameters" element="mapmlns:mapml"/></wsdl:message>
    <wsdl:portType name="MapServiceInterface">
        <wsdl:operation name="getMap">
            <wsdl:input message="tns:mapplDocument"/>
            <wsdl:output message="tns:mapmlDocument"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="MapServiceSoapBinding" type="tns:MapServiceInterface">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getMap">
        <soap:operation soapAction="generalMap" style="document"/>
        <wsdl:input>
          <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
          <soap:body use="literal"/>
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="MapServiceService">
      <wsdl:port name="MapServiceSoapPort" binding="tns:MapServiceSoapBinding">
        <soap:address location="http://pcquaddmg.informatik.uni-stuttgart.de:8081/erspase/seife"/>
      </wsdl:port>
    </wsdl:service>
    <plnk:partnerLinkType name="MapService">
      <plnk:role name="MapServiceProvider">
        <plnk:portType name="tns:MapServiceInterface"/>
      </plnk:role>
    </plnk:partnerLinkType>
</wsdl:definitions>
```

Listing B.2: Federation.wsdl

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions name="webfederationservice"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:tns="http://www.nexus.uni-stuttgart.de/1.0/webfederationservice/"
                  xmlns:awqlns="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
                  xmlns:awmlns="http://www.nexus.uni-stuttgart.de/2.0/AWML"
                  xmlns:crlns="http://www.nexus.uni-stuttgart.de/2.0/CRL"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webfederationservice/"
                  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webfederationservice/"
                xmlns:awqlns="http://www.nexus.uni-stuttgart.de/2.0/AWQL">
      <xsd:import namespace="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
                  schemaLocation="http://w3studi.informatik.uni-stuttgart.de/~kaczmapr/nexus/NexusAwqlSchema.xsd"/>
    </xsd:schema>
    <xsd:schema targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webfederationservice/"
                xmlns:awmlns="http://www.nexus.uni-stuttgart.de/2.0/AWML">
      <xsd:import namespace="http://www.nexus.uni-stuttgart.de/2.0/AWML"
                  schemaLocation="http://w3studi.informatik.uni-stuttgart.de/~kaczmapr/nexus/NexusAwmlSchema.xsd"/>
    </xsd:schema>
    <xsd:schema targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webfederationservice/"
                xmlns:crlns="http://www.nexus.uni-stuttgart.de/2.0/CRL">
      <xsd:import namespace="http://www.nexus.uni-stuttgart.de/2.0/CRL"
                  schemaLocation="http://w3studi.informatik.uni-stuttgart.de/~kaczmapr/nexus/NexusCrlSchema.xsd"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="awqlDocument"><wsdl:part name="parameters" element="awqlns:awql"/></wsdl:message>
  <wsdl:message name="awmlDocument"><wsdl:part name="parameters" element="awmlns:awml"/></wsdl:message>
  <wsdl:message name="crlDocument"><wsdl:part name="parameters" element="crlns:changereport"/></wsdl:message>
  <wsdl:portType name="FederationServiceInterface">
    <wsdl:operation name="queryContextData">
      <wsdl:input message="tns:awqlDocument"/>
      <wsdl:output message="tns:awmlDocument"/>
    </wsdl:operation>
    <wsdl:operation name="insertContextData">
      <wsdl:input message="tns:awmlDocument"/>
      <wsdl:output message="tns:crlDocument"/>
    </wsdl:operation>
    <wsdl:operation name="modifyContextData">
      <wsdl:input message="tns:awqlDocument"/>
      <wsdl:output message="tns:crlDocument"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="FederationServiceSoapBinding" type="tns:FederationServiceInterface">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="queryContextData">
      <soap:operation soapAction="query" style="document"/>
      <wsdl:input><soap:body use="literal"/></wsdl:input>
      <wsdl:output><soap:body use="literal"/></wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="insertContextData">
      <soap:operation soapAction="insert" style="document"/>
      <wsdl:input><soap:body use="literal"/></wsdl:input>
      <wsdl:output><soap:body use="literal"/></wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="modifyContextData">
      <soap:operation  soapAction="modify" style="document"/>
      <wsdl:input><soap:body use="literal"/></wsdl:input>
      <wsdl:output><soap:body use="literal"/></wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="FederationServiceService">
    <wsdl:port name="FederationServiceSoapPort" binding="tns:FederationServiceSoapBinding">
      <soap:address location="http://aspc22.informatik.uni-stuttgart.de:8080/federation/federation"/>
    </wsdl:port>
  </wsdl:service>
  <plnk:partnerLinkType name="FederationService">
    <plnk:role name="FederationServiceProvider">
      <plnk:portType name="tns:FederationServiceInterface"/>
    </plnk:role>
  </plnk:partnerLinkType>
</wsdl:definitions>
```

Listing B.3: EventService.wsdl

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions name="webeventservice" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:tns="http://www.nexus.uni-stuttgart.de/1.0/webeventservice/"
                  xmlns:erlns="http://www.nexus.uni-stuttgart.de/1.0/ERL"
                  xmlns:enlns="http://www.nexus.uni-stuttgart.de/1.0/ENL"
                  xmlns:einfons="http://www.nexus.uni-stuttgart.de/1.0/EventInfo"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
                  targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webeventservice/"
                  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webeventservice/">
      <xsd:import namespace="http://www.nexus.uni-stuttgart.de/1.0/ERL"
        schemaLocation="http://w3studi.informatik.uni-stuttgart.de/~kaczmapr/nexus/EventRegistrationLanguage.xsd"/>
    </xsd:schema>
    <xsd:schema targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webeventservice/">
      <xsd:import namespace="http://www.nexus.uni-stuttgart.de/1.0/ENL"
        schemaLocation="http://w3studi.informatik.uni-stuttgart.de/~kaczmapr/nexus/EventNotificationLanguage.xsd"/>
    </xsd:schema>
    <xsd:schema targetNamespace="http://www.nexus.uni-stuttgart.de/1.0/webeventservice/">
      <xsd:import namespace="http://www.nexus.uni-stuttgart.de/1.0/EventInfo"
        schemaLocation="http://w3studi.informatik.uni-stuttgart.de/~kaczmapr/nexus/EventInfo.xsd"/>
    </xsd:schema>
    <xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified"
                targetNamespace="http://schemas.xmlsoap.org/wsdl/">
      <xsd:import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
                  schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/addressing/"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="StartHeader">
    <wsdl:part name="MessageID" element="wsa:MessageID"/>
    <wsdl:part name="ReplyTo" element="wsa:ReplyTo"/>
  </wsdl:message>
  <wsdl:message name="ContinueHeader"><wsdl:part name="RelatesTo" element="wsa:RelatesTo"/></wsdl:message>
  <wsdl:message name="erlDocument"><wsdl:part name="parameters" element="erlns:predicate"/></wsdl:message>
  <wsdl:message name="enlDocument"><wsdl:part name="parameters" element="enlns:notification"/></wsdl:message>
  <wsdl:message name="eventInfoDocument"><wsdl:part name="parameters" element="einfons:EventInfo"/></wsdl:message>
  <wsdl:portType name="EventServiceInterface">
    <wsdl:operation name="registerEventOperation">
      <wsdl:input message="tns:erlDocument"/>
      <wsdl:output message="tns:eventInfoDocument"/>
    </wsdl:operation>
    <wsdl:operation name="deregisterEventOperation">
      <wsdl:input message="tns:eventInfoDocument"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="EventServiceCallbackPortType">
    <wsdl:operation name="receiveNotification">
      <wsdl:input message="tns:enlDocument"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="EventServiceSoapBinding" type="tns:EventServiceInterface">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="registerEventOperation">
      <soap:operation soapAction="register" style="document"/>
      <wsdl:input>
        <soap:header message="tns:StartHeader" part="MessageID" use="literal" wsdl:required="false"/>
        <soap:header message="tns:StartHeader" part="ReplyTo" use="literal" wsdl:required="false"/>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output><soap:body use="literal"/></wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="deregisterEventOperation">
      <soap:operation  soapAction="deregister" style="document"/>
      <wsdl:input><soap:body use="literal"/></wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="EventServiceCallbackSOAPBinding" type="tns:EventServiceCallbackPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="receiveNotification">
      <soap:operation  soapAction="receiveNotification" style="document"/>
      <wsdl:input>
        <soap:header message="tns:ContinueHeader" part="RelatesTo" use="literal" wsdl:required="true"/>
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="EventServiceService">
    <wsdl:port name="EventServiceSoapPort" binding="tns:EventServiceSoapBinding">
      <soap:address
        location="http://aspc22.informatik.uni-stuttgart.de:8080/axisNexus/services/WebEventService?wsdl"/>
    </wsdl:port>
  </wsdl:service>
  <wsdl:service name="EventServiceCallback">
    <wsdl:port name="EventCallbackPort" binding="tns:EventServiceCallbackSOAPBinding"></wsdl:port>
  </wsdl:service>
  <plnk:partnerLinkType name="EventService">
    <plnk:role name="EventServiceProvider"><plnk:portType name="tns:EventServiceInterface"/></plnk:role>
    <plnk:role name="EventServiceRequester"><plnk:portType name="tns:EventServiceCallbackPortType"/></plnk:role>
  </plnk:partnerLinkType>
</wsdl:definitions>
```

Listing B.4: ContextQuery.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ContextQuery"
             targetNamespace="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQuery"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:client="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQuery"
             xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
             xmlns:awql="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
             xmlns:awml="http://www.nexus.uni-stuttgart.de/2.0/AWML">
    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    TYPE DEFINITION - List of services participating in this BPEL process
    The default output of the BPEL designer uses strings as input and
    output to the BPEL Process. But you can define or import any XML
    Schema type and us them as part of the message types.
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <types>
      <schema attributeFormDefault="qualified" elementFormDefault="qualified"
              targetNamespace="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQuery"
              xmlns="http://www.w3.org/2001/XMLSchema"
              xmlns:awql="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
              xmlns:awml="http://www.nexus.uni-stuttgart.de/2.0/AWML">
        <import namespace="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
          schemaLocation="http://w3studi.informatik.uni-stuttgart.de/~kaczmapr/nexus/NexusAwqlSchema.xsd"/>
        <import namespace="http://www.nexus.uni-stuttgart.de/2.0/AWML"
          schemaLocation="http://w3studi.informatik.uni-stuttgart.de/~kaczmapr/nexus/NexusAwmlSchema.xsd"/>
      </schema>
    </types>
    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    MESSAGE TYPE DEFINITION - Definition of the message types used as
    part of the port type defintions
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <message name="ContextQueryRequestMessage">
        <part name="payload" element="awql:awql"/>
    </message>
    <message name="ContextQueryResponseMessage">
        <part name="payload" element="awml:awml"/>
    </message>
    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    PORT TYPE DEFINITION - A port type groups a set of operations into
    a logical service unit.
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <!-- portType implemented by the ContextQuery BPEL process -->
    <portType name="ContextQuery">
        <operation name="processContextQuery">
            <input  message="client:ContextQueryRequestMessage" />
            <output message="client:ContextQueryResponseMessage"/>
        </operation>
    </portType>
    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    PARTNER LINK TYPE DEFINITION
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <plnk:partnerLinkType name="ContextQuery">
        <plnk:role name="ContextQueryProvider">
            <plnk:portType name="client:ContextQuery"/>
        </plnk:role>
    </plnk:partnerLinkType>
</definitions>
```

Listing B.5: ContextQueryExclude.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ContextQueryExclude"
             targetNamespace="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQueryExclude"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:client="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQueryExclude"
             xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
             xmlns:awql="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
             xmlns:awml="http://www.nexus.uni-stuttgart.de/2.0/AWML">
    <!-- ~~~
```

C BPEL Prozesse

In diesem Anhang werden die BPEL Beschreibungen für die *Integrationsprozesse ContextQuery* und *ContextQueryExclude* exemplarisch aufgeführt. Auf diese Beschreibungen wird in Beispielen aus der Implementierung in Abschnitt 4.4 verwiesen.

Listing C.1: ContextQuery.bpel

```
<process name="ContextQuery"
         targetNamespace="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQuery"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20"
         xmlns:ns4="http://www.nexus.uni-stuttgart.de/2.0/AWML" xmlns:ns1="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
         xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap" xmlns:ns3="http://www.nexus.uni-stuttgart.de/1.0/NSAS"
         xmlns:ns5="http://www.nexus.uni-stuttgart.de/1.0/webfederationservice/"
         xmlns:ns2="http://www.nexus.uni-stuttgart.de/1.0/NSAT"
         xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
         xmlns:client="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQuery"
         xmlns:ora="http://schemas.oracle.com/xpath/extension"
         xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc">
  <partnerLinks>
    <partnerLink name="client" partnerLinkType="client:ContextQuery" myRole="ContextQueryProvider"/>
    <partnerLink name="Federation" partnerRole="FederationServiceProvider" partnerLinkType="ns5:FederationService"/>
  </partnerLinks>
  <variables>
    <variable name="inputVariable" messageType="client:ContextQueryRequestMessage"/>
    <variable name="outputVariable" messageType="client:ContextQueryResponseMessage"/>
    <variable name="invokeContextQuery_queryContextData_InputVariable" messageType="ns5:awqlDocument"/>
    <variable name="invokeContextQuery_queryContextData_OutputVariable" messageType="ns5:awmlDocument"/>
  </variables>
  <sequence name="main">
   <receive name="receiveInput" partnerLink="client" portType="client:ContextQuery"
            operation="processContextQuery" variable="inputVariable" createInstance="yes"/>
   <assign name="AssignInputToAWQL">
      <copy>
        <from variable="inputVariable" part="payload" query="/ns1:awql"/>
        <to variable="invokeContextQuery_queryContextData_InputVariable" part="parameters" query="/ns1:awql"/>
      </copy>
    </assign>
    <invoke name="invokeContextQuery" partnerLink="Federation" portType="ns5:FederationServiceInterface"
            operation="queryContextData" inputVariable="invokeContextQuery_queryContextData_InputVariable"
            outputVariable="invokeContextQuery_queryContextData_OutputVariable"/>
    <assign name="AssignAWMLToOutput">
      <copy>
        <from variable="invokeContextQuery_queryContextData_OutputVariable" part="parameters" query="/ns4:awml"/>
        <to variable="outputVariable" part="payload" query="/ns4:awml"/>
      </copy>
    </assign>
    <reply name="replyOutput" partnerLink="client" portType="client:ContextQuery"
           operation="processContextQuery" variable="outputVariable"/>
  </sequence>
</process>
```

Listing C.2: ContextQueryExclude.bpel

```
<process name="ContextQueryExclude"
         targetNamespace="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQueryExclude"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20"
         xmlns:ns4="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQuery"
         xmlns:ns1="http://www.nexus.uni-stuttgart.de/2.0/AWQL" xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
         xmlns:ns3="http://www.nexus.uni-stuttgart.de/1.0/NSAS" xmlns:ns2="http://www.nexus.uni-stuttgart.de/2.0/AWML"
         xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
         xmlns:client="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQueryExclude"
         xmlns:ora="http://schemas.oracle.com/xpath/extension"
         xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc">
  <partnerLinks>
    <partnerLink name="client" partnerLinkType="client:ContextQueryExclude" myRole="ContextQueryExcludeProvider"/>
    <partnerLink name="ContextQuery" partnerRole="ContextQueryProvider" partnerLinkType="ns4:ContextQuery"/>
  </partnerLinks>
  <variables>
    <variable name="inputVariable" messageType="client:ContextQueryExcludeRequestMessage"/>
    <variable name="outputVariable" messageType="client:ContextQueryExcludeResponseMessage"/>
    <variable name="invokeContextQuery_processContextQuery_InputVariable" messageType="ns4:awqlDocument"/>
    <variable name="invokeContextQuery_processContextQuery_OutputVariable" messageType="ns4:awmlDocument"/>
  </variables>
  <sequence name="main">
    <receive name="receiveInput" partnerLink="client" portType="client:ContextQueryExclude"
             operation="processContextQueryExclude" variable="inputVariable" createInstance="yes"/>
    <assign name="TransformQueryExcludeToAWQL">
      <copy>
        <from expression="ora:processXSLT('ContextQueryExclude.xsl',bpws:getVariableData('inputVariable','payload'))"/>
        <to variable="invokeContextQuery_processContextQuery_InputVariable" part="parameters"/>
      </copy>
      <bpelx:annotation>
        <bpelx:pattern>transformation
        </bpelx:pattern>
      </bpelx:annotation>
    </assign>
    <invoke name="invokeContextQuery" partnerLink="ContextQuery" portType="ns4:ContextQuery"
            operation="processContextQuery" inputVariable="invokeContextQuery_processContextQuery_InputVariable"
            outputVariable="invokeContextQuery_processContextQuery_OutputVariable"/>
    <assign name="AssignAWMLToOutput">
      <copy>
        <from variable="invokeContextQuery_processContextQuery_OutputVariable" part="parameters" query="/ns2:awml"/>
        <to variable="outputVariable" part="payload" query="/ns2:awml"/>
      </copy>
    </assign>
    <reply name="replyOutput" partnerLink="client" portType="client:ContextQueryExclude"
           operation="processContextQueryExclude" variable="outputVariable"/>
  </sequence>
</process>
```

D XSLT Stylesheets

In diesem Anhang werden die XSLT Beschreibungen für die *Integrationsprozesse ContextQuery-Exclude* und *ContextEventOnEnterArea* exemplarisch aufgeführt. Auf diese Beschreibungen wird in Beispielen aus der Implementierung in Abschnitt 4.4 verwiesen.

Listing D.1: ContextQueryExclude.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet
    version="1.0" xmlns:awql="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
    xmlns:awml="http://www.nexus.uni-stuttgart.de/2.0/AWML"
    xmlns:nsas="http://www.nexus.uni-stuttgart.de/1.0/NSAS"
    xmlns:nsat="http://www.nexus.uni-stuttgart.de/1.0/NSAT"
    xmlns:nscs="http://www.nexus.uni-stuttgart.de/1.0/NSCS"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:client="http://nexus.informatik.uni-stuttgart.de/Federation/ContextQueryExclude"
    xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
    xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20"
    xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ora="http://schemas.oracle.com/xpath/extension"
    xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc"
    exclude-result-prefixes="xsl ldap xp20 bpws ora client orcl">
  <xsl:template match="/">
    <awql:awql>
      <xsl:apply-templates/>
      <xsl:if test="client:ContextQueryExcludeProcessRequest/awql:target">
        <xsl:for-each select="client:ContextQueryExcludeProcessRequest/awql:target">
          <awql:exclude><awql:target><xsl:value-of select="."/></awql:target></awql:exclude>
        </xsl:for-each>
      </xsl:if>
    </awql:awql>
  </xsl:template>
  <xsl:template match="*">
  <!-- Elements which are not handled by match functions will be ignored-->
  </xsl:template>
  <xsl:template match="client:ContextQueryExcludeProcessRequest"><xsl:apply-templates/></xsl:template>
  <xsl:template match="awql:aa">
    <xsl:if test="string-length() != 0"><awql:aa><xsl:value-of select="."/></awql:aa></xsl:if>
  </xsl:template>
  <xsl:template match="awql:ecs">
    <xsl:if test="string-length() != 0"><awql:ecs><xsl:value-of select="."/></awql:ecs></xsl:if>
  </xsl:template>
  <xsl:template match="awql:srs">
    <xsl:if test="string-length() != 0"><awql:srs><xsl:value-of select="."/></awql:srs></xsl:if>
  </xsl:template>
  <xsl:template match="awql:geoDataFormat">
    <xsl:if test="string-length() != 0">
      <awql:geoDataFormat><xsl:value-of select="."/></awql:geoDataFormat>
    </xsl:if>
  </xsl:template>
 <xsl:template match="awql:restriction">
   <xsl:if test="*"><awql:restriction><xsl:copy-of select="*"/></awql:restriction></xsl:if>
 </xsl:template>
 <xsl:template match="awql:nearestNeighbor">
   <xsl:if test="*"><awql:nearestNeighbor><xsl:copy-of select="*"/></awql:nearestNeighbor></xsl:if>
 </xsl:template>
</xsl:stylesheet>
```

Listing D.2: ContextEventOnEnterArea.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet
    version="1.0" xmlns:awql="http://www.nexus.uni-stuttgart.de/2.0/AWQL"
    xmlns:awml="http://www.nexus.uni-stuttgart.de/2.0/AWML"
    xmlns:erl="http://www.nexus.uni-stuttgart.de/1.0/ERL"
    xmlns:enl="http://www.nexus.uni-stuttgart.de/1.0/ENL"
    xmlns:nsas="http://www.nexus.uni-stuttgart.de/1.0/NSAS"
    xmlns:nsat="http://www.nexus.uni-stuttgart.de/1.0/NSAT"
    xmlns:nscs="http://www.nexus.uni-stuttgart.de/1.0/NSCS"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:client="http://nexus.informatik.uni-stuttgart.de/EventService/ContextEventOnEnterArea"
    xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
    xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20"
    xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ora="http://schemas.oracle.com/xpath/extension"
    xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc"
    exclude-result-prefixes="xsl ldap xp20 bpws client ora orcl">
  <xsl:template match="/">
    <erl:predicate>
      <erl:templateID>
      nexus:http://www.nexus.uni-stuttgart.de/1.0/events|urn:OnEnterArea|0x01000401f36911d8bcb7e0ab3e91ad69
      </erl:templateID>
      <erl:name>
        <xsl:value-of select="/client:ContextEventOnEnterAreaProcessRequest/client:name"/>
      </erl:name>
      <erl:thresholdProbability>
        <xsl:value-of  select="/client:ContextEventOnEnterAreaProcessRequest/client:thresholdProbability"/>
      </erl:thresholdProbability>
      <erl:parameterList>
      <erl:Parameter>
        <erl:name>object identifier</erl:name>
        <erl:type>de.uni_stuttgart.nexus.types.basic.ObjectId</erl:type>
        <erl:value>
          <xsl:value-of  select="/client:ContextEventOnEnterAreaProcessRequest/client:enteringObject"/>
        </erl:value>
      </erl:Parameter>
      <erl:Parameter>
        <erl:name>observation area</erl:name>
        <erl:type>de.uni_stuttgart.nexus.types.location.Area</erl:type>
        <erl:value>
          <xsl:value-of select="/client:ContextEventOnEnterAreaProcessRequest/client:area"/>
        </erl:value>
      </erl:Parameter>
      <erl:Parameter>
        <erl:name>required accuracy</erl:name>
        <erl:type>java.lang.Double</erl:type>
        <erl:value>
          <xsl:value-of  select="/client:ContextEventOnEnterAreaProcessRequest/client:accuracy"/>
        </erl:value>
      </erl:Parameter>
    </erl:parameterList>
    <erl:predicateManagementParameterList>
      <erl:predicateManagementParameter>
        <erl:name>registration interval</erl:name>
        <erl:type>java.lang.Long</erl:type>
        <erl:value unit="seconds">
          <xsl:value-of select="/client:ContextEventOnEnterAreaProcessRequest/client:registrationDuration"/>
        </erl:value>
      </erl:predicateManagementParameter>
    </erl:predicateManagementParameterList>
    <erl:predicateEvaluationParameterList>
      <erl:predicateEvaluationParameter>
        <erl:name>blocking interval</erl:name>
        <erl:type>java.lang.Long</erl:type>
        <erl:value>0</erl:value>
      </erl:predicateEvaluationParameter>
    </erl:predicateEvaluationParameterList>
    </erl:predicate>
  </xsl:template>
</xsl:stylesheet>
```