Thema:

Serviceorientierte Architektur und Webservices

Studienarbeit

Arbeitsgruppe Wirtschaftsinformatik

Themensteller: Prof. Dr. Hans-Knud Arndt

Betreuer: Prof. Dr. Hans-Knud Arndt

Vorgelegt von: Oliver König

Abgabetermin: 27.03.08

Inhaltsverzeichnis

Verzeichnis der Abkürzungen und Akronyme

Apache AXIS	Apache eXtensible Interaction System
API	Application Programming Interface
DaVincy QWP	DaVincy Quality web-based Platform
EJB	Enterprise Java Beans
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IT	Informationstechnik
J2EE	Java 2 Enterprise Edition
Java EE	Java Platform, Enterprise Edition
JSP	Java Server Pages
SMTP	Simple Mail Transfer Protocol
SOA	Serviceorientierte Architektur
TCP	Transmission Control Protocol
UDDI	Universal Description, Discovery and Integration
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	ohne Beschränkung der Allgemeinheit

Abbildungsverzeichnis

1 Motivation

Ein wesentlicher Erfolgsfaktor von Unternehmen bildet die Geschwindigkeit, mit der es sich in seinem Handeln, an sich stetig wandelnde Marktbedingungen anzupassen vermag. Diesbezüglich gilt es insbesondere die sogenannte Time to Market, also die Zeitspanne von der Produktentwicklung bis zur eigentlichen Platzierung des Produkts am Markt, möglichst gering zu halten, und sich so mögliche Wettbewerbsvorteile zu sichern.

Unter anderem aus dieser Motivation heraus, versuchen Unternehmen im allgemeinen ihre geschäftliche Tätigkeit als Prozesse zu identifizieren, und diese in dokumentierter Form zu strukturieren und zu verwalten. Auf der einen Seite erfolgt hier eine systematische Untergliederung und Dokumentation der einzelnen Prozesse über ihre Teilprozesse und Prozessschritte bis hin zu ihren atomaren Einheiten, den Arbeitsvorgängen. Gleichermaßen werden die Prozesse aber auch untereinander strukturiert, und so in einen gemeinsamen Kontext gebracht, um bestehende Abhängigkeiten zu verdeutlichen. So lassen sich Schwachstellen in den Prozessen und Prozessketten leichter aufdecken, und Verbesserungsmaßnahmen gezielter umsetzten. Letztendlich bringt diese umfassende systematische Strukturierung des unternehmerischen Handelns eine erhöhte Flexibilität bezüglich der Erstellung oder Modifikation von Geschäftsprozessen mit sich.

In der heutigen Zeit stützten sich Geschäftsprozesse zum Teil erheblich auf die Unterstützung der Informationstechnik. Die hierfür eingesetzten Systeme entstanden zumeist als Lösungsansätze für spezifische Probleme oder als Anforderungen für einzelne Unternehmensbereiche. Auch wenn diese Insellösungen im Laufe der Zeit bereichsübergreifend miteinander verbunden wurden, folgen die daraus entstandenen Integrationslösungen im allgemeinen keinem einheitlichen Strukturierungskonzept. Vielmehr sind sie aus situationsbedingten Anpassungsmaßnahmen gewachsen und ihre Teilsysteme und Komponenten wurden zumeist für einzelne Geschäftsbereiche optimiert. So findet die prozessorientierte Ausrichtung von Unternehmen in den eigenen IT-Systemen oft ihre Grenze. Die Umsetzung geforderter Anpassungen kann an dieser Stelle oft nicht zeitnah, oder nur unter unwirtschaftlichen Ressourceneinsatz realisiert werden.

Die Serviceorientierte Architektur setzt hier an und versucht, anhand einer prozessorientierten Strukturierung von Systemen, dem entgegenzuwirken. Im Rahmen dieser Arbeit soll gezeigt werden, durch welche Konzepte dieses verfolgt wird, und wie diese technisch umgesetzt werden können. Hierzu soll unter Anderem auch auf Webservices eingegangen werden, deren Technologien häufig in einem Umfeld von Serviceorientierten Architekturen genutzt werden. Zuletzt soll die Anwendung dieser Technologien durch die Lösung einer praktischen Aufgabe nahegelegt werden.

2 Serviceorientierte Architekt

Die Serviceorientierte Architektur (SOA) beschreibt einen Ansatz der Informationstechnik (IT) für die Architektur von Informationssystemen. Innerhalb einer SOA werden Anwendungsfunktionalitäten gekapselt, als wiederverwendbare Dienste strukturiert und verwaltet, sowie zur Suche und Nutzung in einem Netzwerk bereitgestellt (vgl. Melzer (2007), S. 7).

Die Suche und Nutzung dieser Dienste erfolgt dabei fast ausschließlich von Applikationen und anderen Diensten, und unterliegt bestimmten Prinzipien sowie Standardisierungen. Mit diesen soll eine automatisierte Diensteinbindung zur Laufzeit ermöglicht werden, die möglichst wenig Abhängigkeiten von den Plattformen und Programmiersprachen der beteiligten Einheiten aufweist. Damit soll erreicht werden, dass sich einmal erstellte Dienste, unabhängig von ihrer Implementierung kombinieren und zu komplexeren, sich wiederum anbietenden Diensten strukturieren lassen.

Der Schwerpunkt bei der Erstellung oder auch Änderung von verteilten Anwendungen verlagert sich in serviceorientierten Umgebungen so immer mehr auf die Modellierung von bestehenden Komponenten. Im Idealfall sollten sich auf diese Weise ganze Geschäftsprozesse aus einem bestehenden Satz von Diensten abbilden lassen (vgl. Melzer (2007), S.10). Eine SOA ermöglicht so eine funktionale Zerlegung von Anwendungen, und forciert bei ihrer Konzeption eine prozessorientierte Sichtweise. Unter diesem Gesichtspunkt wird auch der grundlegende Anstoß für die Entwicklung der SOA ersichtlich. Dieser findet sich vor dem Hintergrund wieder, dass bei notwendigen Anpassungen von Geschäftsprozessen, mit der fehlenden Flexibilität betroffenerer IT-Systeme oft hohe wirtschaftliche Einbußen verbunden werden müssen (vgl. Josuttis, (2007), S.1).

Der Begriff der SOA wird bis heute mit keiner allgemeingültigen Definition verbunden. Dennoch lässt sich in der Literatur eine überwiegende Übereinstimmung bestimmter Prinzipien finden, die eine SOA prägen. Diese werden im folgenden Abschnitt beschrieben, bevor auf weitere Merkmale und Aspekte eingegangen werden soll.

2.1 Prinzipien

Eine der grundlegenden Prinzipien der SOA besteht in der Bereitstellung von Funktionalitäten durch Dienste. Diese Dienste unterliegen dabei ähnlichen Anforderungen, wie sie aus der Objektorientierten Programmierung bekannt sind. So stellt ein Dienst in einer SOA eine in sich technisch abgeschlossene Einheit dar (Kapselung), die eigenständig genutzt und so durch mehrer Stellen verwendet werden kann (Wiederverwendung) (vgl. Josuttis (2007), S. 16 f.).

In Bezug auf die Dienste, lässt mit dem Begriff der Interoperabilität ein weiteres wichtiges Prinzip identifizieren. Mit ihr wird im allgemeinen die Fähigkeit der Gegenseitigen Nutzung von heterogenen Systemen bezeichnet. Mit einer SOA wird versucht, einen möglichst hohen Grad an Interoperabilität zwischen den Diensten zu erreichen. Dieses bedeutet vor allem, dass die Interaktion zwischen den Diensten weitestgehend unabhängig von den zu Grunde liegenden Programmiersprachen und Plattformen erfolgt.

Das wohl bedeutendste Prinzip der SOA stellt die Lose Kopplung dar. Mit dem Grad der Kopplung wird im allgemeinen die Abhängigkeit von einzelnen Teilsystemen innerhalb eines Systems gemessen. Eine Lose Kopplung zwischen den Diensten ermöglicht es, dass einzelne Dienste, die zu einem System zusammengeführt wurden, bei Bedarf ausgetauscht oder neu kombiniert werden können (vgl. Mathas (2007), S. 21).

Ein Zeichen der Losen Kopplung lässt sich in der Verwendung von Nachrichtenprotokollen finden (vgl. Mathas (2007), S. 23). Demnach kommunizieren Dienste in einer SOA nicht direkt miteinander, sondern Verpacken ihre Anfragen bzw. Antworten in Nachrichten. Für die Nutzung eines Dienstes, benötigt es hierbei gewisse Kommunikationsvereinbarungen, wie z.B. zu verwendende Zeichensätze oder Übertragungsprotokolle. Des weiteren muss hierzu in Erfahrung gebracht werden, mit welchen Nutzdaten die Nachrichten anzureichern sind, um den Dienstaufruf zu starten. Diese Anforderungen müssen in einem standardisierten, maschinenlesbaren Format als sogenannter *Service Contract* bereitgestellt werden(vgl. Mathas (2007), S. 46).

2.2 Kommunikationsframework

2.2.1 Rollen

Im Rahmen einer SOA werden bezüglich der Funktion die eine beteiligte Anwendung ausüben kann, zwischen den Rollen Dienstanbieter, Dienstverzeichnis sowie Dienstnutzer unterschieden (vgl. Melzer (2007), S. 12).

Ein Dienstanbieter ist eine Anwendung, die zumindest einen Dienst über das Netzwerk zur Nutzung bereitstellt. Hierbei muss sie die bereitgestellte Funktionalität nicht zwingend selbst implementieren, und kann die Dienste anderer Dienstanbieter integrieren und wiederum bereitstellen.

Das Dienstverzeichnis stellt in einer SOA eine zentrale Anlaufstelle zum Registrieren von Diensten aus Sicht der Dienstanbieter, sowie zum Suchen und Finden von Diensten aus Sicht der konsumierenden Einheiten dar. Die Nutzung von Dienstverzeichnissen

sollte sich, um die Komplexität der Kommunikation im Rahmen einer SOA nicht unnötig zu erhöhen, als gewöhnlicher Dienstaufruf gestalten. Die Anzahl der eingesetzten Instanzen von Dienstverzeichnissen gilt in einem SOA-System nicht als beschränkt. Einzelne Instanzen für bestimmte Bereiche oder Dienstkategorien liegen hier nahe (vgl. Melzer (2007), S. 15).

Der Dienstnutzer stellt die konsumierende Einheit in einer SOA dar. Im Unterschied zum Client der klassischen Client-Server-Architektur, ist bei ihm die Adressierung der dienenden Einheit nicht explizit kodiert. Über standardisierte Methoden erfährt der Dienstnutzer von einem geeigneten Dienstanbieter erst zur Laufzeit.

2.3 Aktionen

Um eine standardisierte Kommunikation zwischen Dienstanbieter, Dienstnutzer und Dienstverzeichnis zu gewährleisten, beschränken sie sich in ihrem Handlungsspielraum auf eine definierte Menge von Aktionen. Diese besteht aus dem Veröffentlichen, dem Suchen und dem Binden von Diensten (vgl. Melzer (2007), S. 12).

Das Veröffentlichen eines Dienstes erfolgt durch die entsprechende Registrierung in einem Verzeichnisdienst. Die hier hinterlegten Informationen setzten sich aus einer formalen Beschreibung des Dienstes sowie der Zugriffsmöglichkeiten zusammen. Die veröffentlichten Dienste stehen so für potentielle Dienstnutzer zur Suche bereit. Wurde bei einem Dienstverzeichnis eine passender Eintrag eines Dienstanbieters gefunden, so müssen im weiteren Verlauf die spezifischen Kommunikationsvereinbarungen untersucht werden. Lässt sich eine Einigung gemäß des Service Contracts finden, so gilt der Dienstnutzer als erfolgreich an den Dienst des Dienstanbieters gebunden. Abb. 2.1 zeigt diesen Ablauf.

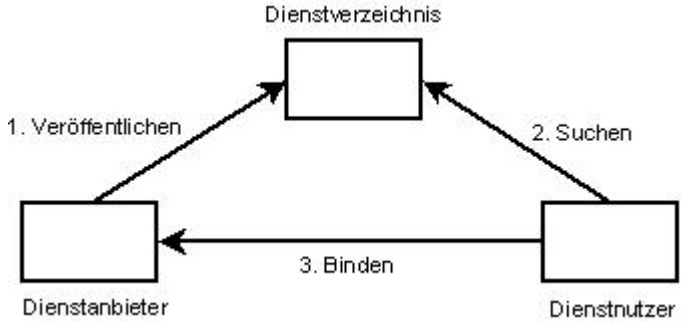

Abb. 2.1: Interaktionen in einer SOA

2.4 Mediation-Systeme

Die elementaren Grundbausteine werden in einem SOA-System durch die Dienste gebildet. Gemäß der Prinzipien der SOA, sollte deren Beziehung zu anderen Diensten auf einer Losen Kopplung beruhen. Sie lassen sich demnach als autonome Komponenten beschreiben, die ihre Funktionalitäten mittels einer geeigneten Schnittstelle zur Verfügung stellen. Diese wiederum wird im Service Contract beschrieben, und findet sich zur Suche in zentralen Registern wieder. Nun benötigt es jedoch noch einer vermittelnden Instanz, die zum einen die Kommunikation zwischen den Diensten ermöglicht, und sie zum anderen in gemeinsame Kontexte setzt. Hier kommen die sogenannten Mediation-Systeme zum Einsatz.

Versucht ein Dienstnutzer eine Interaktion mit einem Dienstanbieter zu starten, so sendet er die aufrufende Nachricht an ein solches Mediation-System. Dieses ermittelt dann durch eine Auswertung der im Dienstverzeichniss hinterlegten Service Contracts ggf. einen geeigneten Dienstanbieter. Hierbei muss das Mediation-System auch nichtfunktionale Anforderungen wie Authentifizierungs-, Transaktions- oder Sicherheitsaspekte mit in Betracht ziehen. Um die Interoperabilität zwischen den beiden Diensten sicherzustellen, müssen zusätzlich weitere Eigenschaften ermittelt werden. Beruhen die Schnittstellen der Dienste beispielsweise auf unterschiedlichen Technologien, so ist es Aufgabe des Mediation-Systems, etwaige Nachrichtentransformationsregeln aus dem Service Contract in Erfahrung zu bringen, und diese bei der Nachrichtenvermittlung anzuwenden. Letztendlich könnten an dieser Stelle auch Abläufe oder Prozesse beschrieben sein, die bei der Ausführung des untersuchten Dienstanbieters einzuhalten sind. Das Heranziehen und die Verfolgung dieser Kontexte, zählt dann zu den Aufgaben eines Mediation-Systems (vgl. Mathas (2007), S. 64).

Mediation-Systeme werden aufgrund ihrer zentralen Bedeutung häufig als Herzstück eines SOA-Systems bezeichnet. Oft wird ihre vermittelnde Funktion zwischen den Diensten mit der von den IP-Routern des Internets verglichen. Welche genauen Aufgaben Mediation-Systeme übernehmen, ist vor allen Dingen von der Konzeption und Ausrichtung des vorliegenden SOA-Systems abhängig. Ist dieses beispielsweise eher als Integrationslösung für heterogene Systeme ausgelegt, so muss es möglicherweise eine Vielzahl von unterschiedlichen Schnittstellen unterstützen. In anderen Fällen könnte beispielsweise die Ablaufsteuerung eines Prozesses, Sicherheit oder Transaktionen im Vordergrund stehen. Letztendlich existieren verschiedene Varianten von Mediation-Systemen, welche mehr oder weniger befähigt sind, bestimmte Aufgaben zu erfüllen. SOA-Systeme sind dabei nicht auf einzelne Varianten beschränkt und ein kombinierter Einsatz ist ebenfalls möglich. Abb. 2.2 zeigt ein solches Szenario.

Mediation-Systeme lassen sich aufgrund ihrer Eigenschaften in verschiedene Varianten unterschiedlicher Komplexität unterscheiden (vgl. Mathas (2007), S.102 ff.). Zu ihnen gehören:

- *Webservice-Engines*, welche oft in Szenarien der Systemintegration und vor allem bei der Nachrichtenvermittlung über das Internet Verwendung finden.
- *SOA-Gateways*, dessen Einsatzschwerpunkt in der Regel in der Integration von datenbanklastigen Alt-Systemen liegt.
- Der *Enterprise Service Bus (ESB),* welches das derzeitig am höchsten entwickelte Mediation-System darstellt, und sich vor allem durch nicht funktionale Merkmale wie Ausfallsicherheit, Transaktionsunterstützung, eine flexible Schnittstellenunterstützung und Nachrichtentransformationsfähigkeit auszeichnet, und somit für die Umsetzung von Geschäftsprozessen optimiert ist.

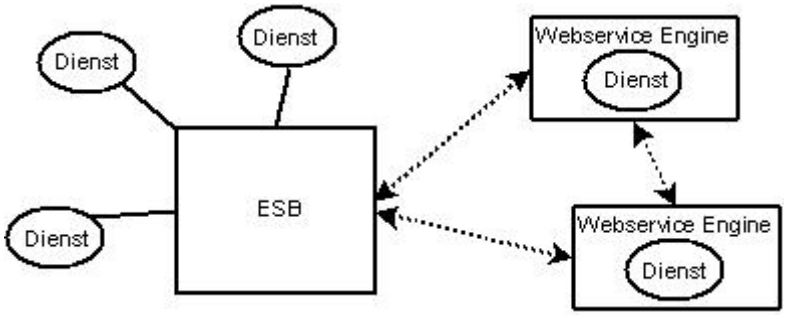

Abb. 2.2: Interaktion unterschiedlicher Medation-Systeme

3 Webservices

3.1 Definition

Der Begriff Webservice lässt sich bis heute mit keiner allgemein anerkannten Definition verbinden. Auch wenn sich in der Literatur viele Abgrenzungen finden lassen, so fallen diese doch recht unterschiedlich aus. Hier legen einige Definitionen Webservices auf konkrete Technologien fest, andere sprechen letztere gar nicht erst an. MATHAS bemerkt zudem: „Web Services und SOA-Systeme werden heute noch häufig als Synonyme betrachtet.“, (Mathas (2007), S. 181; SOA intern).

Im Rahmen dieser Arbeit sollen die Begriffe SOA und Webservices als klar von einander abgrenzt gelten. Ferner soll der Begriff Webservices durch keine konkreten Technologien eingrenzt werden. Es soll vielmehr verdeutlicht werden, dass bestimmte, derzeit verfügbare Technologien, sich für die Bereitstellung von Funktionalität über das Netzwerk, mit Bedacht auf die Interoperabilität und der Losen Kopplung, besonders eignen. Eine Definition, die dieser Haltung entspricht lautet:

„Ein Webservice ist eine über ein Netzwerk zugängliche Schnittstelle zu Anwendungsfunktionen, die mit Hilfe von Standardtechniken des Internets realisiert wird.“, (Snell et al. (2002), S.1; Webservice-Programmierung mit SOAP).

3.2 Stapel der Webservice-Techniken

Eine Möglichkeit die Architektur von Webservices zu beschreiben bietet der Stapel der Webservice-Techniken. Dieser stellt ein Modell dar, welches den technologischen Aufbau von Webservice anhand einer Reihe von Techniken beschreibt, die in aufeinander aufbauenden Schichten organisiert sind. Diese Techniken werden angewandt, um eine dynamische Einbindung von heterogenen, verteilten Anwendungen zu ermöglichen. Nicht jede auf Webservice basierende Integration unterliegt dabei der Bearbeitung aller Schichten. So werden in der Praxis zumeist nur diejenigen Teile implementiert, die den Anforderungen des aktuellen Projekts genügen (vgl. Snell et al. (2002), S. 6 ff.).

Ferner ordnet das Modell den Schichten die jeweils eingesetzten Technologien oder Protokolle zu. Der Stapel der Webservice-Techniken stellt dabei kein endgültiges Modell dar. Im Laufe des technologischen Fortschritts, könnten zugeordnete Technologien ersetzt werden. Wandelnde Anforderungen könnten zu Anpassungen der Schichten führen (vgl. Cerami (2002), S.10).

Im Folgenden werden die Aufgaben der jeweiligen Techniken beschrieben, und die Technologien angeführt, die zur Bewältigung der Aufgaben eingesetzt werden. Eine Beschreibung einer speziellen Auswahl der Technologien soll dann in 3.3 gegeben werden.

Transport: Die Transportschicht regelt die Datenübertragung über ein Netzwerk. Bei der Implementierung eines Webservice unterliegen dem Entwickler hier beinahe keine Einschränkungen in der Wahl der Protokolle, da sich Webservices nahezu auf jedes Transportprotokoll aufsetzen lassen (vgl. Snell et al. (2002), S.9).

Verpackung: Eine Anforderung an Webservices besteht darin, eine Interaktion zwischen heterogenen Systemen zu ermöglichen. Die Aufgabe die sich hieraus ergibt ist, programmiersprachenabhängige Datenstrukturen in ein neutrales Verpackungsformat zu überführen, welches durch alle beteiligten Anwendungen sinngemäß verarbeitet werden kann. Davon ausgehend machen sich Webservices einige besondere Eigenschaften der Extensible Markup Language(XML) im Zusammenhang mit XML-Schema zu nutze. Hierzu benutzen Webservices das Nachrichtenprotokoll SOAP um Nutzdaten für die Übertragung über eine Netzwerk zu verpacken bzw. zu entpacken.

Beschreibung: Eine Anwendung, die einen Webservice nutzen möchte, benötigt zum einen Informationen über die zu verwendenden Protokolle. Zum anderen muss sie wissen, über welche Adresse und mit welchen Aufrufen, die Funktionalität des Webservice zu erwecken ist. Diese Informationen werden als Service Contract mittels der Web Service Description Language(WSDL) beschrieben, und in einem Netzwerk bereitgestellt.

Entdeckung: Die Entdeckungsschicht befasst sich mit der zentralen Speicherung von Dienstbeschreibungen, um potentiellen Nutzern eine Anlaufstelle zu bieten, benötigte Dienste zu suchen und zu finden. Ein weit verbreiteter Entdeckungsmechanismus in diesem Zusammenhang lässt sich in dem Projekt Universal Description, Discovery, and Integration finden(UDDI).

3.3 Eingesetzte Technologien

3.3.1 XML und XML-Schema

Die Extensible Markup Language ist eine vom World Wide Web Consortium(W3C) spezifizierte Auszeichnungssprache. Als solche dient sie der Beschreibung von Daten und stellt einen Rahmenwerk bereit, welches es ermöglicht, Festlegungen zur Verarbeitung dieser Daten zu formalisieren. Da eine XML-Datei textbasiert ist, und dadurch eine

technologieneunabhängige Verarbeitung ermöglicht, eignet sich XML besonders für den Datenaustausch in heterogenen Umgebungen.

Für das Definieren von Strukturen und Inhalten in XML-Dokumenten kommen sogenannte Schemasprachen zum Einsatz. XML Schema, welche selbst eine XML-Anwendung darstellt, gilt in diesem Zusammenhang als Empfehlung des W3C (vgl. Wyke, Watt (2002), S. 5). Ähnlich wie in objektorientierten Programmiersprachen, werden dem Entwickler bei der Arbeit mit XML-Schema zum einen elementare Datentypen in Form von vorbereiteten Datentypisierungen zur Verfügung gestellt. Zum anderen wird ihm anhand einer Menge von sogenannten Schemakomponenten die Möglichkeit gegeben, eigene Datentypen festzulegen. Auf die so erzeugten Typdefinitionen wird dann beim Einsatz der entsprechenden Instanzen verwiesen.

Hält ein XML-Dokument alle XML-Regeln ein (wohlgeformtes XML), und ist es zudem durch eine entsprechende Typdefinition qualifiziert, deren Einschränkung mit der verwendeten Instanz eingehalten wird, so ist dieses Dokument als gültig anzusehen(vgl. W3C (2002)).

Ein solches gültiges Dokument enthält also Verweise auf die von ihm verwendeten Typen und beschreibt somit die eigene semantische Struktur. Dieses ermöglicht eine maschinelle Verarbeitung mittels sogenannter XML-Parser. Jene führen zunächst eine Strukturvalidierung durch und prüfen auf Wohlgeformtheit. Fällt diese positiv aus, kann die Datenvalidation durchgeführt werden, bei der die Inhalte gemäß der referenzierten Typdefinition auf Korrektheit überprüft werden. (vgl. van der Vlist (2003), S. 2).

Angesichts der hier beschriebenen Aspekte, liegt durch den kombiniertem Einsatz von XML und XML-Schema ein Instrumentarium vor, welches bei der Kommunikation in heterogenen, verteilten Anwendungen, eine Typensicherheit gewährleisten kann..

3.3.2 SOAP

Der Nachrichtenaustausch mittels XML und XML-Schema gewährt wie bereits beschrieben, eine Grundlage für die Kommunikation zwischen heterogenen Systemen. Sender sowie Empfänger, unabhängig von Bindungen an Betriebsystem und Programmiersprache, sind in der Lage die gleichen XML-Nachrichten zu verarbeiten, und haben das gleiche Verständnis über die angegebenen Datenstrukturen. Um diese jedoch auch in einen gemeinsamen Kontext zu setzen, müssen Vereinbarungen sowohl über den Aufbau der Nachricht, als auch über die Bedeutung der gesendeten Inhalte getroffen werden (Wang et al. (2007), S. 45).

SOAP (ursprünglich für Simple Object Access Protocol) stellt ein Protokoll für den Nachrichtenaustausch zwischen verteilten Systemen dar. Seine Spezifikation definiert den Aufbau und das Format von Nachrichten. SOAP selbst ist eine Anwendung der XML-Spezifikation und verlässt sich auf Standards wie XML-Schema (Snell et al. (2002), S. 13)

Eine Bindung an ein bestimmtes Trägerprotokoll wird nicht durch die SOAP-Spezifikation festgelegt. Aufgrund seiner weiten Verbreitung sowie der Tatsache, dass es von Firewalls im allgemeinen nicht geblockt wird, wird diesbezüglich in der Praxis zumeist auf das Hyptertext Transfer Protocol (HTTP) in Verbindung mit dem Transmission Control Protocol (TCP) zurückgegriffen (Wang et al. (2007), S. 42).

Eine SOAP-Nachricht ist ein XML-Dokument, dessen Wurzelelement durch den SOAP-Envelope repräsentiert wird. Jener muss laut seiner Spezifikation genau einen SOAP-Body besitzen, dem optional ein SOAP-Header vorangestellt werden kann. Die eigentlichen Nutzdaten sind hierbei im Body-Element zu strukturieren. Im Header-Element sind Angaben zum Routing, Transaktionskontexte oder sicherheitsrelevante Informationen anzugeben. Die SOAP-Spezifikation selbst macht bezüglich der Inhalte der beiden Elemente keine Vorgaben, beschränkt ihr Format jedoch auf gültiges XML..

Der folgende XML-Code stellt eine mögliche Instanz einer SOAP-Nachricht dar. Die Struktur für den Envelope ist hierbei über den XML-Namensraumbezeichner `http://www.w3.org/2001/12/soap-envelope` zu referenzieren. Dieser zeigt auf das entsprechende XML-Schema.

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
    <s:Header>
    </s:Header>
    <s:Body>
    </s:Body>
</s:Envelope>
```

3.3.3 WSDL

Die Web Service Description Language (WSDL) ist eine XML-Spezifikation und dient einem Web Service als Meta-Sprache für die von ihm angebotenen Funktionen und Daten, sowie die verwendeten Datentypen und Austauschprotokolle zu beschreiben, und diese im Zusammenhang mit seiner Netzwerkadresse zu veröffentlichen. Ein Client der

die Dienste eines Web Service in Anspruch nehmen will, hat aufgrund der bereitgestellten Information, die Möglichkeit seinerseits Methoden zu generieren, mit Hilfe derer der beschriebene Web Service dann genutzt werden kann. Eine WSDL stellt als Service Contract somit eine Vorraussetzung für eine dynamische Einbindung von Web Services von Clientapplikationen dar.

4 Entwicklung einer serviceorientierten Schnittstelle

4.1 Einführung

Im Rahmen meines Studiums mit Fachrichtung Wirtschaftsinformatik an der Otto-von-Guerricke Universität Magdeburg, absolvierte ich im Wintersemester 2003/2004, das damit verbundene Berufspraktikum bei der Daimler AG am Standort Esslingen. Dort wurde ich im Bereich der Systemsicherheit der Abteilung RIC/AS zugeordnet. Unter der Projektleitung von Dr. Stefan Greiner war ich an der Entwicklung des analytischen Informationssystems DaVincy Quality web-based Platform (im Folgenden als DaVincy QWP bezeichnet) beteiligt.

Die DaVincy QWP stellte eine Initiative im Rahmen des Qualitätsmanagement der Daimler AG dar. Ziel war es ein Informationssystem mit Analysefähigkeit für die Managementebene zu erstellen. Aufgrund der Auswertung von Kfz-Schadfalldaten, sollten sich so mögliche Schlüsse auf qualitative Schwachstellen in der Produktion der Kraftfahrzeuge ziehen lassen. Die zu Grunde liegenden eingesetzten Schadfalldaten wurden hierfür von mehreren unternehmensinternen. dezentralen Stellen bezogen. Diese wiederum, hatten direkte Anbindungen an die Systeme der Daimler-Vertragswerkstätte.

Geplant war es, schon während der Entwicklung des Systems, die Datenbasis mit den bis zu dem damaligen Zeitpunkt verfügbaren Schadfalldaten anzureichern. Festgelegt wurde dazu, dass lediglich Schadfälle an Fahrzeugen, die nach dem Jahr 1996 produziert wurden importiert werden sollten. So wurde bereits nach dem Entwurf des Datenbank-Schemas und der Implementierung entsprechender Boulk-Import Funktionalität damit angefangen, sukzessiv Schadfalldaten für alle Kfz-Marken und Kfz-Baureihen des Konzerns in das System zu importieren. Bis zur Fertigstellung und während des Betriebs, sollte der Datenbestand dann in periodischen Abständen aktualisiert werden.

Die komplette Steuerung des Systems erfolgte damals über eine Web-Oberfläche, die sowohl für die Administration als auch für die eigentliche Nutzung durch den Endnutzer konzipiert wurde. Zum Zwecke der Analyse, sollte dem Nutzer über seinen Webbrowser die Möglichkeit gegeben werden, unter der individuellen Festlegung von Parametern, verschiedene vom System bereitgestellte Datenbankabfragen anzustoßen, und die entsprechenden Ergebnisse anschließend durch verschiedene Visualisierungstechniken (Tabellen und Grafiken) aufbereitet zu betrachten.

Noch während der Entwicklungsphase wurde das System unter anderem aus Testzwecken, für einen erlesenen Kreis von Interessenten zur Nutzung bereitgestellt. Die daraus hervorgegangenen Rückkopplungen wurden über den Projektleiter direkt an die ent-

sprechenden Entwickler weitergeben. Die einzelnen Reaktionen auf das System ließen sich als durchgehend positiv einordnen, und die gesamte Resonanz lag deutlich über den Erwartungen der Entwickler. Zu diesem Zeitpunkt wurden von Seiten der Nutzer auch immer öfter Anfragen über weitere Zugriffsmöglichkeiten als den Webbrowser gestellt.

> *Diese Thematik nehme ich als Ansatz um unter Berücksichtigung der QWP-Architektur einen Datenbankabfragedienst zu entwickeln, der auf keinen bestimmten Client beschränkt ist.*

Die Softwarearchitektur bereffend, wurde bei der Entwicklung der DaVincy QWP, zu Gunsten eines komponentenorientiertes Modells entschieden. Um genauer zu sein, basierte sie auf der von der Firma SUN herausgegebenen Spezifikation J2EE[1] und verwendete die darüber bereitgestellten APIs. Die Grundlage dafür bildeten ein Websphere Application Server[2] im Zusammenhang mit dem relationalen Datenbankmanagementsystem DB2.

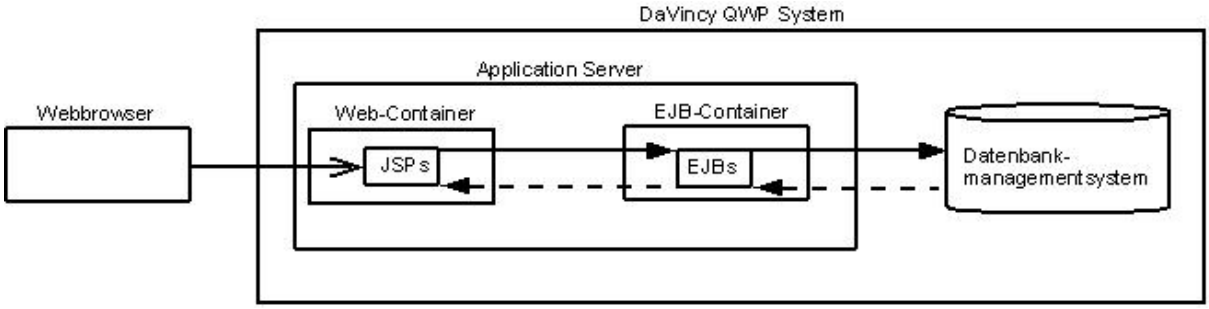

Abb. 4.1: Datenfluss der DaVincy QWP

Die Funktionalität der Geschäftslogik wurde hierbei als Enterprise Java Beans(EJB) implementiert und im entsprechenden EJB-Container des Applikationsserver verwaltet. Der Zugriff von außen erfolgte mittels Webbrowser, welcher auf die vom Web Container des Applikationsserver verwalteten Java Server Pages (JSP) abrief. Abb 4.1 zeigt den allgemeinen Datenfluss auf.

---

[1] Die Java EE 1.3 Spezifikation soll im Rahmen dieser Arbeit nicht erläutert werden und ein grundlegendes Wissen ihr bezüglich, wird im Folgenden vorrausgesetzt. Eine Übersicht der zur Zeit aktuellen Version Java EE 5 ist unter http://java.sun.com/javaee/technologies zu finden.

[2] Der Websphere Application Server stellt in diesem Zusammenhang eine Referenzimplementierung der J2EE dar, und stellt eine entsprechende Laufzeitumgebung zur Verfügung.

4.2 Vorgehen

Um den Entwicklungsprozess des Dienstes überschaubar zu halten, soll an dieser Stelle zunächst das Vorgehen strukturiert und festgehalten werden. Dieses soll im weiteren Verlauf als Vorgehensmodell, und somit als Übersicht und Orientierungshilfe während der Entwicklung dienen. Zu diesem Zweck wird der Entwicklungsprozess in die folgenden Teilabschnitte gegliedert, in denen hier zugleich das Vorgehen zur Übersicht veranschaulicht wird.

Analyse: An erster Stelle soll im Rahmen einer Vorstudie das Umfeld der zu entwickelnden Software, und die eigentliche Problemstellung festgehalten werden. Danach können erste Überlegungen zur Projektplanung angestellt werden. Aufgrund der geringen Komplexität des Projekts soll auf diese jedoch nicht mehr eingegangen werden. Stattdessen wird direkt auf eine Anforderungsanalyse eingegangen, welche die ermittelten Anforderung aus der Vorstudie vervollständigen und spezifizieren soll. Auf Grundlage der Anforderungsspezifikation, soll dann eine Modellierung des Systems vorgenommen werden. Die daraus hervorgehenden Ergebnisse sollen letztendlich eine detaillierte und konsistente Leistungsbeschreibung, und die Grundlage für den Entwurf darstellen.

Entwurf: Ausgehend von den Ergebnissen der Analyse, soll hier die Architektur des Systems festgelegt werden. Hierzu wird das System unter Beachtung von logischen Zusammenhängen in Teilsysteme aufgeteilt, und das Zusammenwirken spezifiziert. Der nächste Schritt wird dann die unmittelbare Vorbereitung für die Implementierung darstellen, indem ausgehend von allen vorherigen Ergebnissen ein Objektentwurf abgeleitet wird.

Implementierung: Während der Implementierung wird der vorliegende Objektentwurf letztendlich in den eigentlichen Programmcode umgesetzt.

4.3 Analyse

4.3.1 Vorstudie

Die DaVincy-QWP stellt ein analytisches Informationssystem dar, für dessen Nutzung ursprünglich ausschließlich Webbrowser vorgesehen waren, und das dementsprechend konzipiert wurde. Der Webbrowser bildete somit die einzige Zugriffsmöglichkeit auf die vorliegende Funktionalität. Im Laufe der Zeit wurden jedoch auch Wünsche auf eine weitere Zugriffsmöglichkeit geäußert. Diese Schnittstelle sollte nicht auf eine spezielle

Art von Client konzipiert sein, sondern vielmehr einen weitestgehend universalen Zugriff erlauben.

Unter der zuvor getroffenen Festlegung gilt es nun mehr ein System zu entwickeln, welches einen plattform- und programmiersprachen-unabhängigen Zugriff auf einen Datenbankabfragedienst ermöglicht. Das gesamte System ist dabei als Komponente für die DaVincy-QWP zu erstellen, und muss sich in die entsprechende Softwarearchitektur integrieren lassen.

4.3.2 Anforderungsanalyse

Aus der Vorstudie werden bereits einige Anforderungen ersichtlich. Diese sollen jetzt unter vollständiger Betrachtung der spezifischen Gegebenheiten, und möglicher Anwendungsfälle ausdrücklich spezifiziert werden.

Bereits erwähnt wurde, dass das System als Komponente der DaVincy-QWP zu erstellen ist. Diese unterliegt dem angesprochene Websphere-Application-Server. Folglich muss es sich in das Komponentenmodell der Java EE Spezifikation einreihen lassen. Ein weiterer wichtiger Aspekt, betrifft die Kompatibilität zur unternehmensinternen Firewall. Die Zugriffe sollen vor auch von außenliegenden Zweigstellen des Konzerns möglich sein. Die Beschränkung auf eine Kommunikation über den Netzwerkport 80 ist damit unumgänglich. Des weiteren sollte herausgestellt werden, dass das System von mehreren Benutzern gleichzeitig nutzbar sein muss. Die Anwendungsmöglichkeiten von außen stellt sich in Abb. 4.2 dar.

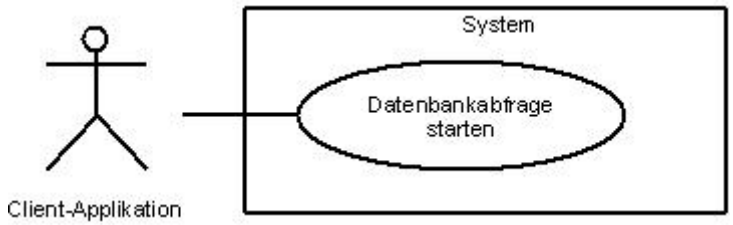

Abb. 4.2: Use-Case-Diagramm

Die Gesamtheit der Anforderungen lassen sich nach ihrer Art unterteilt wie folgt darstellen:

- Funktionale Anforderungen
  - Das System muss den Anstoß der Datenbankabfrage ermöglichen

- Nicht funktionale Anforderungen
  - Die Nutzung des Systems soll unabhängig von der Programmiersprache des Clients sein.
  - Das System soll Mehrbenutzerbetrieb unterstützen.
  - Das System soll über die unternehmensinterne Firewall ansprechbar sein.
  - Das System muss sich in einer Java EE Laufzeitumgebung einsetzen lassen.

4.4 Modellierung

Nachdem die Anforderungen jetzt genau spezifiziert wurden, können ausgehend von ihnen, nun die ersten Überlegungen zur Modellierung des Systems vorgenommen werden. So sollen zunächst teilnehmende Objekte erkannt und klassifiziert werden. Diese können dann in einen zeitlichen Kontext gebracht werden, so dass bereits ereignisgetriebene Zustandsänderungen der Objekte verdeutlicht werden. Die gewonnene Erkenntnisse werden letztendlich in einen funktionalen Zusammenhang gebracht, welcher die Verarbeitungsschritte des Systems anhand des Datenflusses aufzeigt.

Für die Identifizierung der Objekte im Rahmen der Analyse lässt sich folgende Klassifizierung vornehmen(vgl. Brügge (2004), S. 207):

- Entitätsobjekte stellen vom System verwaltete Informationen dar
- Steuerungsobjekte repräsentieren von außen zugreifbare Funktionalität
- Grenzobjekte bilden die von außen zugreifbaren Schnittstellen des Systems

Unter Beachtung der geforderten Interoperabilität ergibt sich das erste Objekt für das System. Gesendete Nachrichten müssen durch eine vermittelnde Instanz in ein programmiersprachen-unabhängiges Format gebracht werden. Umgekehrt müssen die empfangenen Nachrichten von dieser wieder entsprechend umgewandelt werden. Als Grenzobjekt lässt sich somit der Vermittler festhalten. Bezüglich der Steuerung lässt sich das Auslösen der Datenbankabfrage, und somit der Abfragedienst als Steuerungs-

objekt identifizieren. Die Datenbankschnittstelle ist als Entitätsobjekt anzuordnen. Abb. 4.3 zeigt die identifizierten Objekte.

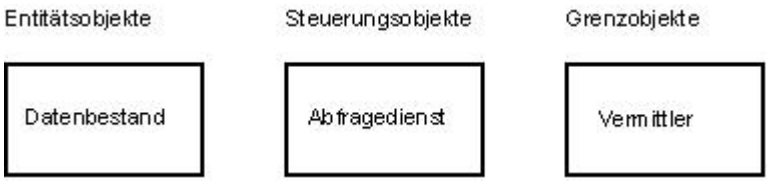

Abb. 4.3: Klassifikation der ermittelten Objekte

Nun sollen die identifizierten Objekte auf ihre Beziehungen untersucht werden. Problematisch ist hierbei, dass der Vermittler selbst ein komplexes Teilsystem darstellt, dessen Eigenschaften an dieser Stelle noch nicht bekannt sind. Letztere sind vor allem von den eingesetzten Kommunikationstechnologien abhängig. Diese wiederum, können erst mit der Konzeption der Systemarchitektur festgelegt werden, wo besonders auf die nicht funktionalen Eigenschaften eingegangen werden soll. Aus diesem Grund wird der Vermittler zunächst als einfache Klasse festgelegt. Mit der Verfolgung dieses Ansatz ergibt sich ein erstes Objektmodell wie es in Abb. 4.4 dargestellt ist.

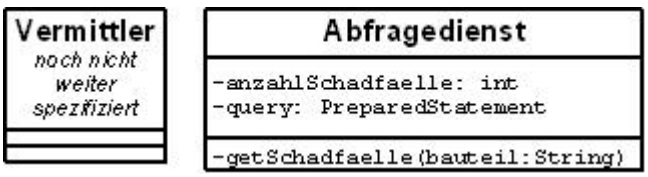

Abb. 4.4: Objektmodell

Bei der Betrachtung des Objektmodells im Zusammenhang mit dem Anwendungsfall, lassen sich für die Objekte keine Zustandsänderungen identifizieren. Aus diesem Grund wird an dieser Stelle auf eine dynamische Modellierung verzichtet.

Um nun auf funktionalen Aspekte des Systems einzugehen wird als Betrachtungsgrundlage der Datenfluss des Systems durchdacht. Für den Anwendungsfall ergibt sich folgendes, in Abb. 4.5 dargestellte Sequenzdiagramm, welches das sogenannte funktionale Modell wiederspiegelt.

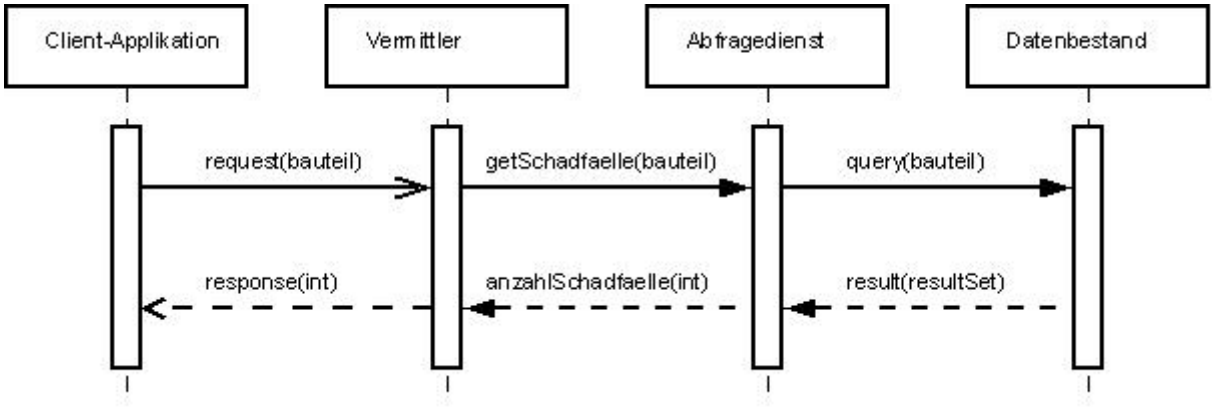

Abb. 4.5: funktionales Modell

4.5 Entwurf

4.5.1 Systementwurf

Nachdem die Analyse nun den Leistungsumfang des Systems spezifiziert hat, kann nun auf die Konzeption der Architektur eingegangen werden. Diese wird dann den Rahmen für die weitere Verfeinerung der Analyseergebnisse, bis hin zur Festlegung und Anordnung der konkreten Datenstrukturen bilden.

Für die Annäherung an eine geeignete Architektur, soll das System zunächst in Teilsysteme mit abgegrenzten Aufgabenbereichen zerlegt werden. Wie bereits in der Modellierung festgestellt wurde, sollte der Zugriffspunkt auf das System ein Vermittler sein. Mit ihm wird zugleich das erste Teilsystem identifiziert. Seine Aufgabe besteht in der Transformation von Nachrichten zwischen System und Client. Sie geht aus der Anforderung der Programmiersprachen-unabhängigen Anbindung hervor. Die anderen nicht funktionalen Anforderungen, müssen ebenfalls in diesem Teilsystem Beachtung finden, da es sie auch auf die Zugriffsart abzielen. Aus ihnen ergeben sich, dass zum einem der Netzwerkport-Port 80 zur Datenübertragung verwenden werden muss. Zum Anderem sollte die Kommunikation asynchron erfolgen, da sich sonst mehrer Clients bei parallelen Zugriffen blockieren könnten.

Eine geeignetes Übertragungsprotokoll bildet in diesem Zusammenhang SOAP, das es nicht nur die genannten Anforderungen erfüllen kann, sondern sich zudem durch seinen hohen Standardisierungsgrad auszeichnet. Aus diesem Grund soll an dieser Stelle SOAP als Nachrichtenprotokoll festgelegt werden, Daraus ergibt sich, dass das die vermittelnde Instanz eine Komponente darstellt, die in der Lage ist SOAP-Nachrichten zu verarbeiten (SOAP-Engine).

Die restlichen Teilsysteme ergeben sich unmittelbar aus der funktionalen Anforderung. Hier stellen sowohl das Verarbeiten und Weiterreichen der Client-Anfrage, als auch die

entsprechende Abfrage der Datenbank eigenständige Aufgaben. Das vollständige Systementwurfsmodell wird in Abb. 4.6 zusammengefasst, und verdeutlicht unter Anderem, in wieweit sich die Teilsysteme untereinander Nutzen.

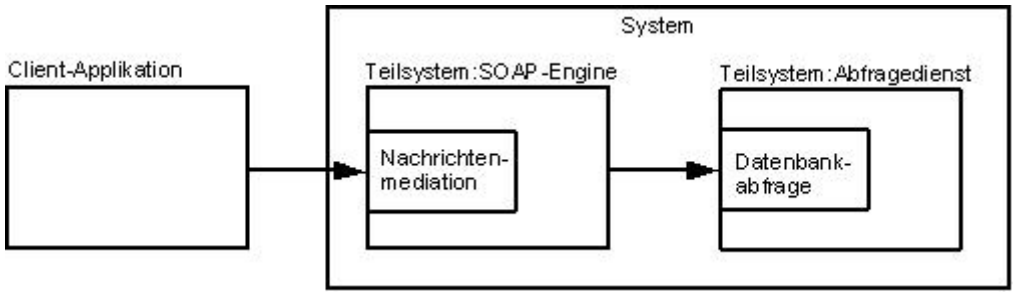

Abb. 4.6: Systementwurfsmodell

4.5.2 Objektentwurf

Im Rahmen der zuletzt konzipierten Systemarchitektur sollen nun die Ergebnisse der Analyse so weit verfeinert werden, dass sie als unmittelbare Grundlage für die Implementierung verwendet werden können. Dazu soll zunächst das ermittelte Objektmodell mit den Ergebnissen des Systementwurfs abgeglichen werden.

Der Vermittler im ursprünglichen Objektmodells wurde im Rahmen des Systementwurfs auf eine SOAP-Engine eingeschränkt. Für einen Client, der den Dienst des Systems nutzen möchte, bedeutet dieses, dass er seine Anfrage als SOAP-Nachricht senden muss, und eine solche als Antwort zu erwarten hat. In dem Zusammenhang muss er jedoch wissen, mit welchen genauen Informationen er die gesendeten SOAP-Nachrichten anreichern muss, und mit welchen genauen Antwortdaten er zu rechnen hat. Aus diesem Grund wird die Bereitstellung eines Service Contracts durch das System notwendig. Für die Nachrichtenübertragung mit SOAP, ist diesbezüglich die WSDL ein häufig genutzter Standard, und soll somit auch hier Verwendung finden.

Bis auf die SOAP-Engine sind nun alle Klassen des Objektmodells soweit spezifiziert, dass sie sich direkt auf Programmcode-Klassen übertragen lassen könnten. SOAP-Engines werden im allgemeinen für einzelne Projekte nicht selbst entwickelt. Aufgrund der Komplexität müsste dies mit sehr hohen Aufwänden verbunden werden. Zudem hat man mit dem Einsatz bewährter Fremdprodukte wohl in den meisten Fällen die effizientere Alternative. Eine populäre SOAP-Engine stellt die frei erhältliche Apache Axis der Apache Software Foundation dar. Da Apache Axis als Implementierung in Java erhält-

lich ist, und sich als Servlet im Web Container einer Java EE Laufzeitumgebung betreiben lässt, soll es in diesem Projekt als SOAP-Engine dienen.

Nun muss das Objektmodell an die Festlegung von Apache Axis als Vermittler angepasst werden. Im Zusammenhang mit dem Einsatz einer WSDL-Instanz, und der Verwendung von Java, bietet Apache Axis bestimmte Werkzeuge an. Diese sollen die Einbindung eines Dienstes erleichtern.. Das Tool WSDL2JAVA generiert hierbei aus einer WSDL-Instanz einen entsprechenden Schnittsellen-Code für den einzubindenen Dienst. Damit ist die Anbindung an Apache-Axis nicht mehr Teil des Objektmodells, und wird unmittelbar in der Implementierung ausgeführt. Abb.4.7 zeigt das aktualisierte Objektmodell.

Abb. 4.7: aktualisiertes Objektmodell

4.6 Implementierung

Aufgrund der Erkenntnisse, die während des Objektentwurfs gemacht wurden, lässt sich das Vorgehen für die Implementierung in die folgenden, aufeinander aufbauenden Schritte untergliedern:

1. Implementierung der Klasse des Objektmodells
2. Erstellen der WSDL-Instanz
3. Genieren des Schnittstellencodes mit WSDL2JAVA
4. Integrieren des Datenbankdienstes in den Schnittstellencode

Abschließend bleibt zu sagen, dass sich für das Projekt die Implementierung eines Webservice, als geeignete Lösung dargestellt hat. Ausschlaggebend hierfür war vor allem, dass sich eine Integration des Systems über das Internet bewerkstelligen lassen sollte. Webservices können Standardtechnologien des Internets nutzen und machen so von den gängigsten Infrastrukturen Gebrauch.

5 Zusammenfassung und Ausblick

Mit dieser Arbeit sollte herausgestellt werden, welche Bedeutung die Serviceorientierte Architektur für Unternehmen einnehmen kann. Ferner wurde versucht die Rolle von Webservices in diesen Zusammenhang einzugliedern. Auf der anderen Seite war es auch ein Ziel, die beiden, prägenden Begriffe dieser Arbeit voneinander anzugrenzen. Die praktische Aufgabe sollte verdeutlichen, dass in heterogenen Umgebungen, die über Unternehmensinfrastrukturen hinauslaufen, die Technologien, die mit Webservices verbunden werden, oft die angemessenen Alternativen zur Integration darstellen.

Literaturverzeichnis

Brügge, B. (2004): Objektorientierte Softwaretechnik: mit UML, Entwurfsmustern und Java. 2. Aufl., München

Cerami, E. (2002): Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. Beijing et al.

Josuttis , N. M., SOA in practice: the art of distributed system design. Sebastopol, CA et al.

Mathas, C. (2007): SOA intern: Praxiswissen zu Service-orientierten IT-Systemen. München – Wien

Melzer, I. (2007): Service-orientierte Architekturen mit Web Services: Konzepte - Standards – Praxis. 2. Aufl., München

Snell, J. (2002): Webservice-Programmierung mit SOAP, Beijing u.a.

van der Vlist, E. (2002): XML Schema: XML-Daten modellieren. Köln u.a.

W3C (2002): Extensible Markup Language (XML) 1.0 (Zweite Auflage): Deutsche Übersetzung. http://www.edition-w3c.de/TR/2000/REC-xml-20001006 27. März 2008

Wang, D. (2004): Java Web Services mit Apache Axis. Frankfurt

Whyke, R. A.; Watt, A. (2002): XML Schema Essentials. New York, NY

Abschließende Erklärung

Ich versichere hiermit, dass ich die vorliegende Studienarbeit selbständig, ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Magdeburg, den 27. März 2008