UNIVERSITÄT HAMBURG, FACHBEREICH INFORMATIK,
ARBEITSBEREICH SOFTWARETECHNIK

DIPLOMARBEIT

Analyse und Restrukturierung eines bestehenden Software-Systems zur Integration in eine Service orientierte Architektur

vorgelegt von:
Guido Wilken

Matrikel-Nr.:
5300049

Email:
0wilken@informatik.uni-hamburg.de

Betreuer:
Prof. Dr. Heinz Züllighoven

Zweit-Betreuer:
Dr. Wolf-Gideon Bleek

28. Februar 2009

Abstract

Das CommSy stellt als gewachsene Web-Applikation, die zunächst nicht auf die Verwendung von Diensten hin konzipiert wurde, eine gute Grundlage zur Untersuchung von Problemen dar, die sich bei Integration einer bestehenden Applikation in eine SOA ergeben können. Der Vermutung, dass die Voraussetzungen zur Einführung von Web Services nicht gegeben sind, wird zunächst eine Analyse der Architektur nachgehen, um mögliche Konflikte zu identifizieren. Ziel der Arbeit soll es sein, durch Implementierung verschiedener Web Services die Grundlage für die SOA-Fähigkeit der CommSy Web-Applikation zu legen. Um die Zustandslosigkeit der Dienste zu gewährleisten, wird eine Abgrenzung der Funktionalität vorgenommen, die sinnvoll mit Hilfe von Web Services angeboten werden kann. Dabei wird berücksichtigt, welche Informationen, die das CommSy bereitstellt, einen Mehrwert bieten, wenn Sie über alternative Clients zugänglich sind.

„Well, I'll eat it," said Alice, „and if it makes me grow larger, I can reach the key; and if it makes me grow smaller, I can creep under the door: so either way I'll get into the garden, and I don't care which happens!" ...

Lewis Carroll

Danksagung

Ich möchte mich bei allen bedanken, die mir diese Arbeit ermöglicht haben. Insbesondere ist dies Wolf-Gideon Bleek, der als mein Betreuer Zeit und Mühe in nicht unerheblichem Umfang zu dieser Arbeit beigesteuert haben.

Meiner Schwester und meinen Eltern möchte ich an dieser Stelle für Ihre großartige Unterstützung bei der Wahrung der neuen deutschen Rechtschreibung und dem damit verbundenen häufigen Lesen der Arbeit danken.

Des Weiteren möchte ich mich bei allen Freunden und Bekannten bedanken, die mir alle, auf die ein oder andere Weise, in regelmäßigen Abständen neue Kraft gegeben haben, wenn es einmal schwierig wurde.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Des Weiteren erkläre ich hiermit, dass ich mit der Einstellung dieser Diplomarbeit in den Bestand der Bibliotheken der Universität Hamburg einverstanden bin.

Guido Wilken

Hamburg, den 28. Februar 2009

Inhaltsverzeichnis

Kapitel 1

1 Einleitung

Die Integration einer bestehenden Web-Applikation in eine SOA bringt einige Herausforderungen mit sich. Das CommSy stellt als gewachsene Web-Applikation, die zunächst nicht auf die Abbildung von Prozessen hin konzipiert wurde, eine gute Grundlage zur Untersuchung von Problemen dar, die sich bei einer solchen Integration ergeben können.

1.1 Problemstellung

1.1.1 Erweiterung einer Web-Applikation um SOA-Fähigkeit

Die Konzeption einer SOA geht im Wesentlichen davon aus, dass die Betrachtung der zu implementierenden Funktionalität nicht aus der technischen Perspektive oder mit dem Blick auf vorhandene und zu bearbeitende Resourcen geschieht, sondern Geschäftsprozesse im Vordergrund stehen. In diesem Zusammenhang bezieht sich der Begriff der Architektur, der in dem Akronym SOA zu finden ist, mehr auf den Aufbau der gesamten „SOA“, als auf die Architektur eines einzelnen, an der SOA beteiligten Softwaresystems. Für den Entwurf der Dienste, die durch ein beteiligtes System zur Verfügung gestellt werden, stellt dies einen vollständigen Paradigmenwechsel dar. Wird eine SOA implementiert, so zeigt sich häufig bei der Planung der Dienste, die angeboten werden sollen, dass sowohl neue Softwaresysteme entwickelt werden müssen, die bisher nicht abgebildete Prozesse zur Verfügung stellen, als auch bestehende Applikationen in die SOA einzubetten sind. Bei neu zu entwickelnden Applikationen stellt der Paradigmenwechsel kaum große Schwierigkeiten dar. Nachdem umzusetzende Prozesse analysiert wurden, können diese entsprechend implementiert werden. Schwierigkeiten ergeben sich hingegen häufig bei der Einarbeitung bestehender Applikationen, die nicht mit dem Wissen um abzubildende Prozesse implementiert wurden.

1.1.2 Herausforderungen bei der Erweiterung

Die bereits seit geraumer Zeit etablierte Schichtenarchitektur, die bei konsequenter Umsetzung eine Trennung von Funktionalität und Präsentation vorsieht, erleichtert die geplante Integration einer vorhandenen Applikation in eine SOA erheblich. Es gilt, die abzubildenden Prozesse zu analysieren und zu prüfen, ob verwendete Resourcen und Aktionen bereits mit den in der Applikation vorhandenen Resourcen bzw. Aktionen übereinstimmen. In der Regel verfahren derzeitig vorhandene Softwaresysteme nach dem CRUD-Ansatz (Create,

Retrieve, Update, Delete). Stimmen Resourcen der Prozesse und der Applikation überein, so sind die im Idealfall vorhandenen Methoden der Datenzugriffsschicht eine gute Grundlage für die Verwendung der Resourcen in den SOA-Diensten. Eine SOA soll und will aber weit darüber hinaus gehende Dienstleistungen zur Verfügung stellen. Dies beginnt bei der Auffindbarkeit der Dienste einer SOA und endet nicht zuletzt in vielen Anforderungen an einzelne Dienste, die später ausführlich diskutiert werden. Eine flexible Verwendung von Leistungsmerkmalen in anderen Systemen ist nur möglich, wenn diese plattformunabhängig und ohne Kenntnis über die Implementierung zugänglich sind. Ein SOA setzt daher voraus, dass Dienste gekapselt und zustandslos verfügbar sind. Erst auf der Seite der Dienstempfänger oder in orchestrierten Diensten dürfen Zustandsinformationen und Kontextwissen zur Weiterverarbeitung der von einzelnen Diensten generierten Daten genutzt werden. Die architektonischen Anforderungen, die an eine Standalone-Applikation gestellt werden, sind weit geringer, als sie es bei einer SOA sind. Um den neuen Anforderungen gerecht werden zu können, sind ggfs. aufwändige Änderungen der Architektur nötig. Insbesondere betrifft dies in vielen Fällen die Zustandslosigkeit der Dienste. Web-Applikationen setzen in der Regel eine Session voraus, da ihre Nutzung Zustandsinformationen wie beispielsweise Benutzerinformationen benötigt. Bei bestehenden Software-Systemen ist es daher ggfs. nötig, Anpassungen am Datenmodell und an den Prozessen vorzunehmen, um Dienste implementieren zu können, die den Anforderungen einer SOA genügen.

1.1.3 Web Services erfüllen die geforderten Kriterien

Web Services kapseln die Implementierung von Funktionalität und stellen diese mit Hilfe von XML plattformunabhängig zur Verfügung. Da das CommSy bereits über eine konsequent umgesetzte Schichtenarchitektur verfügt, gehe ich davon aus, dass eine Implementierung der Web-Services mit wenig Refactoring der bestehenden Architektur möglich sein wird. Die Diplomarbeit wird sich demnach hauptsächlich mit dem Entwurf der zu implementierenden Web Services beschäftigen. Dabei wird ein wesentlicher Bestandteil der Untersuchung sein, inwieweit Prozesse, die über den einfachen Datenzugriff auf Resourcen innerhalb des CommSy hinaus gehen, abgebildet werden können. Unabhängig von der abgebildeten Funktionialität der implementierten Dienste erscheint die Wahl von Web Services als Technologie gut, da es mit ihnen einfach möglich ist, Web Services über verschiedene Transportschichten zur Verfügung zu stellen. Dies ermöglicht es, einer weiteren zentralen Anforderung an Dienste einer SOA zu genügen. Die Anbindung neuer Clients, die über unterschiedliche Kanäle auf die Web Services zugreifen, kann so mit geringem Aufwand umgesetzt werden. Die Identifikation von minimaler Funktionalität, die sich zur Bereitstellung von SOA-fähigen Diensten eignet, wird es ermöglichen, komplexe Prozesse aufzubrechen. Mit Hilfe der Orchestrierung dieser Dienste lassen sich anschließend aufwändige Prozesse einfacher und verständlicher abbilden.

1.2 Ziel der Arbeit

Ziel der Arbeit soll es sein, durch Implementierung verschiedener Web Services, die Grundlage für die SOA-Fähigkeit der CommSy Web-Applikation zu legen. Die Zugriffsmöglichkeiten auf das System durch verschiedene Clients stellen sich konsistent und einheitlich dar.

Um die Zustandslosigkeit der Dienste zu gewährleisten, wird eine Abgrenzung der Funktionalität vorgenommen, die sinnvoll mit Hilfe von Web Services angeboten werden kann. Dabei wird berücksichtigt, welche Informationen, die das CommSy bereitstellt, einen Mehrwert bieten, wenn sie über alternative Clients zugänglich sind. Des Weiteren soll eine abschließende Untersuchung feststellen, ob und in welchem Umfang Schwierigkeiten bei der Umsetzung der über den bisherigen Ansatz - Schichtenarchitektur und CRUD - hinausgehenden Anforderungen entstehen können.

1.3 Aufbau der Arbeit

Die Arbeit wird sich zunächst mit den nötigen Grundlagen beschäftigen. Technologien und Konzepte aus dem Bereich SOA und Web Services, die in der Implementierung Verwendung finden, werden ausführlich vorgestellt. Dabei werden auch weitere verwandte Technologien wie XML-RPC oder CORBA eingeführt, um eine Begründung für die Auswahl der eingesetzten Technologien zu ermöglichen. Ebenso finden sich im Grundlagenkapitel die Definitionen von Architektur, Analyse von Softwaresystemen und Refactoring. Das Kapitel „Mehrwerte durch SOA“ wird sich anschließend der Hauptthese der Arbeit widmen und Mehrwerte schildern, die durch Implementierung einer SOA gewonnen werden können. Die hier vorgestellten Mehrwerte werden im Kapitel „Planung der Implementierung“ daraufhin untersucht, inwieweit sie im konkreten Fall des CommSy, bedingt durch die Umsetzung, erzielt werden können. Das Kapitel wird die Entwurfsphase der Implementierung widerspiegeln und Erfahrungen wiedergeben, die aus den ersten Prototypen gewonnen werden konnten. Anschließend wird die Vorgehensweise für die Implementierung geplant und auf die spezifischen Technologien, die zum Einsatz kommen, eingegangen. Kapitel „Implementierung der Web Services“ beschreibt die Implementierung der Web Services im CommSy. Nach erfolgreicher Implementierung der Web Services wird eine Analyse der erzielten Ergebnisse vorgenommen. In einer abschließenden Bewertung werden die Mehrwerte, die diese Arbeit in das CommSy einbringen will, untersucht und ein Ausblick darauf gegeben, wie eine weitere Entwicklung unter dem Aspekt einer SOA aussehen könnte.

Kapitel 2

2 Grundlagen

In diesem Kapitel werden die der Diplomarbeit zugrunde liegenden Technologien und Strukturen vorgestellt. Sie werden dabei nur soweit erläutert, wie ihr Verständnis für den hier benötigten Einsatz erforderlich ist. Nicht alle hier vorgestellten Technologien werden in der Implementierung Verwendung finden. Ihre Einführung bietet die Grundlage für die Wahl der in der Implementierung verwendeten Technologien (vgl. Kapitel 4.2, S. 29).

2.1 Architektur

> *The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.*
>
> vgl. Bass et al [BASS et al. 2003]

Mit den extern sichtbaren Eigenschaften der Komponenten sind im Zusammenhang dieser Definition nicht nur deren Schnittstellen gemeint, sondern auch weitere Parameter, wie beispielsweise Performance, Verhalten im Fehlerfall oder Nutzung von verteilten Ressourcen, über die andere Komponenten Annahmen machen können. Diese Definition der Architektur impliziert eine abstrahierende „Blackbox“-Sicht auf ein System oder Programm. Eine Herausforderung bei der Architekturanalyse wie auch bei der Konzeption einer Architektur besteht unter anderem in der Granulierung dieser Abstraktion. Zu starke Abstraktion verbirgt wichtige Informationen, die Fehlentscheidungen hätten vermeiden können, zu geringe Abstraktion hingegen behindert den Überblick.

Der IEEE Standard 610.12-1990 definiert „Architektur“ weniger spezifisch:

> *The organizational structure of a system or component.*
>
> vgl. IEEE 610.12-1990 [iee 1990]

Aber auch hier ist die in [BASS et al. 2003] angesprochene Abstraktion implizit enthalten. Es gibt etliche weitere Definitionsversuche für den Begriff Architektur. Diese beiden seien herausgegriffen, um das Spektrum der Spezifizität dieser Definitionen aufzuzeigen und die Abstraktion und Spezifikation der Komponenten als wesentlichen Teil der Architektur hervorzuheben.

2.1.1 Architekturmuster

Auch im Bereich der Software Architektur haben sich vergleichbar zu den Entwurfsmustern gut funktionierende Muster etabliert, die nicht nur Teilaspekte eines Softwaresystems oder die Lösung wiederkehrender Probleme betreffen, sondern die gesamte Struktur des Systems beschreiben. Häufig verwendete und bekannte Architekturmuster sind beispielsweise „Model View Controller“ oder die „Client-Server“-Architektur. Das im CommSy verwendete Architekturmuster ist eine Schichtenarchitektur. Hier werden einzelne Komponenten des Softwaresystems Schichten zugeordnet, die semantisch zusammengehören. Wie viele Schichten ein System aufweist und welche Funktionen diese erfüllen, ist durch das Architekturmuster selbst nicht vorgegeben. Die Anzahl und Funktionalität der Schichten, die für ein zu entwerfendes Softwaresystem sinnvoll sind, hängt vom Einzelfall ab. Allerdings haben sich auch hier gängige Richtwerte etabliert. Innerhalb von Softwaresystemen, deren Architektur ein Schichtenmodell zu Grunde liegt, findet man demnach häufig folgende drei Schichten:

1. Präsentationsschicht
2. Logikschicht
3. Datenhaltungsschicht

Das wohl populärste und bereits 1983 standardisierte Schichtenmodell mit sieben Schichten ist das OSI-Referenzmodell. Die aktuelle Version ist in der Norm ISO/IEC 74981:1994[1] nachzulesen.

Ist eine Schichtenarchitektur strikt, so dürfen Komponenten einzelner Schichten nur auf Funktionalität darüber- oder darunterliegender Schichten zugreifen. Diese Forderung lässt sich in einigen Fällen nicht aufrechterhalten. Idealerweise sollte aber versucht werden, schichtenübergreifende Kommunikation zu reduzieren um die Kopplung der Komponenten so gering wie möglich zu halten.

2.1.2 Architektursichten

[CLEMENTS et al. 2003] versteht unter Architektursichten die Repräsentation einer Menge von Systemelementen und Beziehungen dieser Elemente zueinander. Dies setzt voraus, dass die Menge der repräsentierten Elemente auf Basis der Repräsentation inhärenter Kriterien ausgewählt wurden. Zu einer „Architektursicht“ gehört demnach ein Standpunkt (bei [CLEMENTS et al. 2003] „viewpoint“), der diese Kriterien vorgibt. Gleichermaßen findet man entsprechende Definitionen auch im IEEE Standard P1471-2000:

> *A view is a representation of a whole system from the perspective of a set of concerns.*
>
> vgl. IEEE P1471-2000 [iee 2000]

> *A viewpoint is a specification of the conventions for constructing and using a view.*
>
> vgl. IEEE P1471-2000 [iee 2000]

[1] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=20269

Welche Sichten relevant sind, hängt stark von Zielsetzung und Kontext der Analyse ab. Dazu haben sich verschiedene Modelle etabliert, die Sichten und die damit verbundenen Standpunkte und Kriterien zusammenfassen, auf die an dieser Stelle allerdings nicht weiter eingegangen wird.

2.2 Analyse von Softwaresystemen

Softwaresysteme lassen sich auf verschiedene Aspekte hin untersuchen. Die Architektur und Qualität des Quelltextes kann dabei genauso Gegenstand der Untersuchung sein wie Skalierbarkeit oder Performance des Softwaresystems unter bestimmten Randbedingungen. Im Rahmen dieser Diplomarbeit werde ich mich bei der Analyse des CommSy auf die Untersuchung der Architektur beschränken. Auch diese Analyse kann unter verschiedenen Gesichtspunkten und vor allem auf unterschiedlichen Abstraktionsebenen stattfinden (vgl. Kapitel 2.1.2, S. 6). Die zur Architekturanalyse des CommSy eingesetzte Software „Sotograph“[2] ermöglicht es, Klassen und Packages, die in den Augen der Entwickler des Sotographen zu fein granuliert für einen Architekturanalyse sind, zu Subsystemen zusammenzufassen. Die Definition der Subsysteme so wie der erlaubten bzw. verbotenen Zugriffe dieser Subsysteme aufeinander stellen die Basis der Analysen dar, die mit Hilfe des Sotographen durchgeführt werden können. Die Schichtenarchitektur des CommSy ist durch die einmalige Zuordnung aller Klassen bzw. Packages einzelner Schichten zu den entsprechenden Subsystemen im Sotographen abgebildet.

2.3 Restrukturierung

Unter Restrukturierung oder auch Refactoring versteht man das Verändern der internen Struktur von Quellcode, ohne sein externes Verhalten zu ändern. Solche Umstrukturierungen werden in der Regel in kleinen Schritten vorgenommen, um die Gefahr von Fehlern zu reduzieren. Eine Vielzahl solcher Veränderungen zusammengenommen dient der Verbesserung des Quellcodes unter verschiedenen Aspekten, ohne eine Veränderung von anderen Bereichen des Quellcodes erforderlich zu machen. Dies kann beispielsweise eine Umstrukturierung einer Methode zur besseren Lesbarkeit oder das Austauschen eines Sortieralgorithmus für einen Geschwindigkeitsgewinn sein. Solche Umstrukturierungen können erheblich zur Verbesserung der Produkt- und Codequalität beitragen, obwohl sie in vielen Fällen nur mit geringem Aufwand verbunden sind. Martin Fowler stellt in seinem Buch „Refactoring. Wie Sie das Design vorhandener Software verbessern“ [FOWLER 2005] einige typische Fehler oder Unsauberkeiten von Code vor, die er „übel riechenden Code“ oder im Englischen „code smells“ nennt. Für diese Beispiele werden Umstrukturierungen vorgeschlagen, die zur Verbesserung des entsprechenden Quellcodes führen. Größere Umstrukturierungen, die eine gesamte Architektur betreffen, sind hingegen ein aufwändiger Prozess, der gute Planung erfordert, um die Funktionalität des Softwaresystems nicht zu beeinträchtigen. Die Restrukturierung von Quellcode sollte immer durch ausführliche Tests unterstützt werden, um die Funktionsfähigkeit des Codes nach der Überarbeitung beweisen zu können.

[2] http://www.software-tomography.ch/html/sotograph.html

2.4 SOA - Service-Oriented Architecture

Erstmalig geprägt wurde der Begriff SOA 1996 durch die Analysten-Gruppe Gartner (vgl. [LIEBHART 2007], S. 6). Neben unzähligen Versprechungen und einer populistischen „Durchseuchung“ des IT-Marktes mit SOA-Gütesiegel, „1-Click-SOA“-Werkzeugen und anderen Fallstricken, hat sich seitdem auch auf der technischen Seite einiges weiter entwickelt. Dieses Kapitel befasst sich mit den theoretischen und technischen Grundlagen, die sich hinter dem Begriff „Service-Oriented Architecture“ verbergen.

2.4.1 Motivation

Die Frage, warum es überhaupt nötig ist, Objekte sprachen- und plattformunabhängig auszutauschen, lässt sich mit nur einem Wort beantworten: Integration. Die zunehmende Komplexität und wachsende Anforderungen an Softwaresysteme führten bereits in frühen Zeiten zu dem Wunsch nach Wiederverwendbarkeit. Beginnend mit Methoden und Prozeduren als kleinste Einheiten, die eine Wiederverwendbarkeit erlauben, hat dieser Wunsch nach Objekt- und Komponentenorientierung zur Einführung von Serviceorientierung geführt. Die Einheiten, die wiederverwendet werden können, sind dabei nicht nur größer geworden, sondern haben auch ihren Einflussbereich erweitert. Waren Methoden und Prozeduren früher, beispielsweise in der Sprache C, nur sehr eingeschränkt wiederverwendbar, so bieten Dienste heute eine plattform- und sprachenunabhängige Wiederverwendbarkeit im Netzwerk. Wiederverwendbare Dienste bilden nicht mehr nur einfache Funktionalität ab, wie das bei Methoden und Prozeduren der Fall war, sondern spiegeln in vielen Fällen ganze Prozesse wieder. Der Fokus einer dienstorientierten Architektur ist nicht mehr nur technischer Natur, sondern richtet sich in der Regel an Geschäftsprozessen aus.

2.4.2 Was bedeutet SOA?

In [LIEBHART 2007] finden sich neben dem Hinweis darauf, dass es keine einheitliche Definition des Begriffs SOA gibt, diverse Beschreibungen und Definitionen verschiedener Hersteller und Analysten. Allen Definitionen sind eine Reihe von Grundsätzen und Komponenten gemeinsam, die hier vorgestellt werden sollen. Zunächst aber sei die Definition der OASIS (Organization for the Advancement of Structured Information Standards) herausgegriffen:

> *A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.*
>
> vgl. OASIS [MACKENZIE et al. 2006], S. 29

Schon diese Definition zeigt, wie viel Freiraum für die Implementierung bleibt und klärt die häufigsten Missverständnisse auf, die SOA mit unterschiedlichen Produkten oder Technologien in Verbindung bringen. Die Einführung des Begriffes SOA geschah mit dem Grundgedanken, Geschäftsprozesse in den Fokus der Betrachtung zu ziehen und die Technik in den Hintergrund zu stellen. Die vom SOA-Ansatz geforderte Dienstorientierung richtet sich demnach primär an Geschäftsprozessen aus. Diese können sowohl durch einzelne Dienste, als auch durch die Komposition mehrerer Dienste abgebildet werden. Die Be-

trachtung der technischen Umsetzung dieser Dienste kommt dann erst an zweiter Stelle. Allen Definitionen und Implementierungen ist der Dienst als zentrale Grundkomponente einer SOA gemeinsam. In allen Implementierungen finden sich Instrumente zur Modellierung der Geschäftsprozesse. Auch das Schichtenmodell einer SOA (Präsentations-Ebene, Orchestrierungs-Ebene, Service-Ebene und Integrations-Ebene) findet sich bei allen Herstellern wieder (vgl. [LIEBHART 2007]).

2.4.3 SOA-Dienst

Einen zentralen Begriff, der im Zusammenhang mit SOA immer wieder fällt, lohnt es sich, genauer zu betrachten. Was bedeutet der Begriff „Dienst“ im SOA-Kontext? Hinter ihm verbirgt sich die zentrale Komponente des SOA-Konzepts, mit der die Brücke zwischen der Perspektive der Geschäftsprozesse und der technischen Perspektive auf die Implementierung geschlagen wird. Eine Reihe von Anforderungen, die an die Dienste gestellt werden (vgl. Kapitel 2.4.5, S. 10), lassen sich sowohl auf technischer Ebene, als auch auf Ebene der Geschäftsprozesse diskutieren. Ausgehend von den Geschäftsprozessen, die in einer SOA abgebildet werden sollen, entspricht ein Dienst einem Prozess oder Teilprozess. Dabei ist zunächst unerheblich, ob dieser Dienst später eigenständig umgesetzt werden kann oder aus der Komposition verschiedener Teilprozesse entsteht. Im Zusammenhang mit der technischen Umsetzung entspricht der Begriff des Dienstes der Implementierung einer bestimmten Funktionalität, die durch eine wohldefinierte Schnittstelle mit Vor- und Nachbedingungen gegeben ist. Auch hier ist nicht entscheidend, ob diese Implementierung andere Dienste nutzt, oder alleine funktionsfähig ist. Interessant an der Idee des Dienstes ist, dass dieser, als zentraler Bestandteil einer SOA, die Kommunikation zwischen der technischen und der Unternehmensperspektive fordert und fördert.

2.4.4 SOA Architektur-Modelle

Im SOA-Umfeld sind im Wesentlichen die drei nachfolgend erläuterten Architektur-Modelle (vgl. [STARKE und TILKOV 2007], S. 27-30) zu finden. Basierend auf den verschiedenen Modellen gibt es entsprechende Frameworks oder Implementierungen für eine SOA. Je nach Einsatzzweck und Beschaffenheit der Daten und Prozesse, die hauptsächlich innerhalb der SOA verarbeitet werden sollen, können die Modelle mehr oder weniger gut und flexibel unterstützen.

Schnittstellenorientiertes Modell

Das schnittstellenorientierte Modell erinnert stark an Prinzipien, die aus der objektorientierten Programmierung bekannt sind. Wie die Methoden von Klassen sich über ihre Signatur definieren, liegt der Fokus bei den Diensten einer SOA hier auf der Schnittstelle. Die Verwendung der Dienste folgt in der Regel dem Request-Response oder dem One-Way MEP (vgl. Kapitel 2.5.2, S. 17), das dem Aufruf von Methoden mit oder ohne Rückgabewert gleicht. CORBA, DCOM oder auch RPC sind Technologien, denen dieses Modell zugrunde liegt. Web Service basierte Implementierungen mit Hilfe von SOAP können ebenfalls als schnittstellenorientiert betrachtet werden. SOAP-Nachrichten werden dann zur Übertragung der Parameter für auszuführende Operationen genutzt. Die Schnittstellendefinitionen

finden sich in der WSDL wieder. Ein Dienst ist im schnittstellenorientierten Modell in der Regel eine semantisch zusammengehörende Menge von Operationen.

Nachrichtenorientiertes Modell

Das nachrichtenorientierte Modell stellt die ausgetauschten Nachrichten und damit die in ihnen enthaltene Information in den Vordergrund. Auch hier kommt bei Web Service basierten Implementierungen in der Regel SOAP zum Einsatz. Die Nachrichtenspezifikationen sind dann nicht an der Signatur einer entsprechenden Operation orientiert, sondern fokussieren die zu übertragende Information. Im Gegensatz zum schnittstellenorientierten Modell bestehen Dienste oft aus nur einer einzelnen oder wenigen Operationen. Häufig wird der Begriff Operation komplett vermieden.

Die Grenze zum schnittstellenorientierten Modell ist unscharf. So können die Informationen, die eine übertragene Nachricht enthält, durchaus zur Ausführung einer zugeordneten Operation führen. Parameter einer Operation, die in Nachrichten eines schnittstellenorientierten Modells übertragen werden, sind aber auf der anderen Seite auch nichts anderes als Informationen.

Ressourcenorientiertes Modell

Das ressourcenorientierte Modell unterscheidet sich stark von den beiden anderen, nah beieinander liegenden Modellen. Im Vordergrund der Betrachtung stehen hier eindeutig identifizierbare Ressourcen. Die einzigen Möglichkeiten der Interaktion mit diesen Ressourcen sind Lesen, Ändern, Löschen und Erzeugen. Diese Operationen müssen zur Abbildung der gesamten Anwendungssemantik genutzt werden. Das ressourcenorientierte Modell wurde 2000 von Fielding in seiner Dissertation [FIELDING 2000] unter dem Namen REST (REpresentational State Transfer) vorgestellt (vgl. Kapitel 2.4.6, S. 13).

2.4.5 Anforderungen an Dienste in einer SOA

Eine der zentralen Anforderungen, die an eine SOA gestellt werden, ist die Interoperabilität. Dabei steht die Unabhängigkeit von Systemplattform und Programmiersprache bei der Kommunikation im Vordergrund. In einer SOA miteinander kommunizierende Systeme sind in der Lage, im Netzwerk verteilte Funktionalität anzubieten, zu entdecken und zu nutzen, ohne deren Implementierung zu kennen (vgl. SOA-Definition OASIS, Kapitel 2.4.2, S.8). Ein Beispiel für eine solche Kommunikation sind nachrichtenbasierte Protokolle. Die dabei ausgetauschten Nachrichten sind definiert und an Vor- und Nachbedingungen gebunden. Für die beteiligten Systemen ist es nur nötig, die Nachrichtenformate des Kommunikationsprotokolls zu kennen, um gewünschte Aktionen anzustoßen oder auszuführen.

Nachfolgend vorgestellte Anforderungen an Dienste sind weitgehend anerkannt und dienen als Grundlage bei der Konzeption einer SOA (vgl. [ERL 2007]).

Standardized Service Contracts

> *Services within the same service inventory are in compliance with the same contract design standards.*

vgl. [ERL 2007], S. 130

Service Loose Coupling

Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment.

vgl. [ERL 2007], S. 168

Service Abstraction

Service contracts only contain essential information and information about services is limited to what is published in service contracts.

vgl. [ERL 2007], S. 215

Dienste möglichst abstrakt zu entwerfen bedeutet, so wenig Details über die Implementierung des Dienstes preiszugeben wie möglich. Dies ermöglich nicht nur, die darunterliegende Implementierung auszutauschen oder schrittweise zu erneuern (siehe Bild 2.1), sondern unterstützt auch beim Entwurf von lose gekoppelten Diensten.

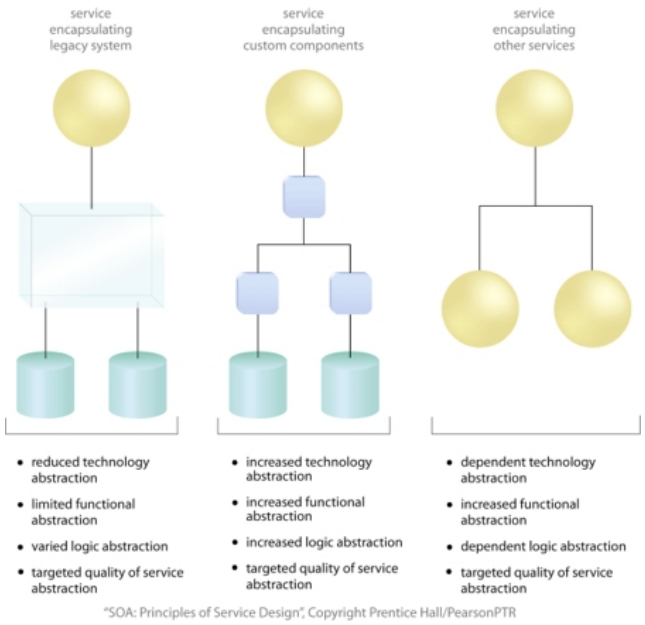

Abbildung 2.1: verschiedene Stufen der Abstraktion, [ERL 2007], S. 236

Service Reusability

Services contain and express agnostic logic and can be positioned as reusable enterprise resources.

vgl. [ERL 2007], S. 259

Um eine SOA sinnvoll zu implementieren ist es bei der Identifizierung und dem Design von bereitzustellenden Diensten wichtig, ihre Wiederverwertbarkeit zu berücksichtigen. Dienste innerhalb einer SOA, die so speziell oder komplex sind, dass sie nur von einer Applikation oder innerhalb eines einzigen Prozesses vorkommen, sind nutzlos im Sinne einer SOA. Der Wert einer SOA ergibt sich zu einem großen Teil erst dadurch, dass Dienste an vielen Stellen zum Einsatz kommen können.

Service Autonomy

> *Services exercise a high level of control over their underlying runtime execution environment.*
>
> vgl. [ERL 2007], S. 296

Das Prinzip der Service Autonomie fordert eine hinreichende Kontrolle des Dienstes über seine Ausführungsumgebung, damit er die angebotenen Leistungen performant und zuverlässig erfüllen kann. Die gemeinsame Nutzung von Ressourcen, die ein Dienst benötigt oder Mehrfachnutzung der physikalischen Systeme, auf denen ein Dienst läuft, können die Zuverlässigkeit dieses Dienstes signifikant beeinträchtigen.

Service Statelessness

> *Services minimize resource consumption by deferring the management of state information when necessary.*
>
> vgl. [ERL 2007], S. 333

Da Zustandsinformationen innerhalb von Diensten zu einer signifikanten Belastung der Systemressourcen führen können, wird in der Regel gefordert, Dienste zustandslos zu implementieren. Sollte dies abhängig von der Beschaffenheit des Dienstes nicht möglich sein, wird in [ERL 2007] gefordert, die Zustandsinformationen auszulagern.

Service Discoverability

> *Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.*
>
> vgl. [ERL 2007], S. 369

Dieses Designprinzip beschränkt sich nicht nur auf die zentrale Fähigkeit einer SOA, automatisiertes Auffinden und Einbinden von Diensten über ein Repository zu ermöglichen. Vielmehr gilt es zusätzlich, angebotene Dienste mit ausreichend Meta-Daten anzureichern, so dass auch ein Mensch einen gewünschten Dienst leicht im Repository finden kann. Bei steigender Anzahl von Diensten, die in einer SOA zur Verfügung stehen, wird die Auffindbarkeit von Diensten immer wichtiger.

Service Composability

> *Services are effective composition participants, regardless of the size and complexity of the composition.*
>
> vgl. [ERL 2007], S. 393

Ein wesentlicher Vorteil der Komponierbarkeit von Diensten besteht in der Möglichkeit, komplexe Problemstellungen auf Teilprobleme abzubilden. Es gilt dabei zu beachten, dass sich Dienste auch als Teile von komplexen Kompositionen so performant wie möglich verhalten sollten.

2.4.6 Technologien aus dem SOA-Umfeld

Da die architektonischen Anforderungen, die an eine SOA gestellt werden, nicht an spezifische Technologien gebunden sind, haben sich im Laufe der Zeit verschiedene Implementierungsmöglichkeiten entwickelt, die mit unterschiedlichem Fokus den Aufbau einer SOA unterstützen können. An dieser Stelle werden die bekanntesten Vertreter vorgestellt und ihre Merkmale gegenüber den anderen Möglichkeiten abgegrenzt.

BPEL

Mit Hilfe der Business Process Execution Language ist man in der Lage, Prozesse innerhalb einer SOA zu modellieren. Zumeist mit grafischen BPEL-Editoren wird ein Prozess aus den verschiedenen zur Verfügung stehenden Diensten zusammengestellt. Ein solcher Prozess lässt sich dann innerhalb einer BPEL-Engine ausführen. BPEL-Prozesse sind selbst wieder Dienste einer SOA und stellen ihr Interfacebeschreibung als WSDL zur Verfügung. Sie lassen sich dadurch leicht in andere Prozesse einbinden und die schrittweise Umsetzung komplexer Abläufe wird stark vereinfacht. Da eine entsprechende Implementierung für das CommSy im Rahmen dieser Diplomarbeit nicht vorgesehen ist, wird hier nicht weiter auf die Details von BPEL eingegangen. Eine ausführliche Abhandlung findet sich beispielsweise in [BLANVALET et al. 2006].

REST - REpresentational State Transfer

REST wurde 2000 von Roy Thomas Fielding in seiner Dissertation „Architectural Styles and the Design of Network-based Software Architectures“ vorgestellt (vgl. [FIELDING 2000]). REpresentational State Transfer ist ein Architektur Modell, das, im Gegensatz zu nachrichten- oder schnittstellenbasierten Modellen, den Fokus auf zustandsbehaftete Objekte legt. Auf diese kann über einen eindeutigen Identifizierer zugegriffen werden. Die einzigen auf den Ressourcen erlaubten Operationen sind Lesen, Ändern, Löschen und Anlegen. In seiner Dissertation schlägt Fielding für Implementierung von REST-Anwendungen HTTP mit den dazugehörigen Methoden (GET, PUT, DELETE und POST) vor. Eine REST Applikation lässt sich so leicht mit Hilfe eines Webservers implementieren. Andere Implementierungen dieses Modells sind aber denkbar. Vorteile beim Einsatz von REST sind die Einfachheit und die Verwendung von seit langem etablierten Standards wie HTTP als Transportprotokoll und URIs für die Adressierung. Das ressourcenorientierte Modell eignet sich für eine Vielzahl von Anwendungsfällen, in denen andere Ansätze zu erheblichem

Overhead führen würden. Komplexere Workflows lassen sich damit aber nur schwer abbilden, Prozessbeschreibungen oder Dienstorchestrierung, wie sie beispielsweise BPEL bietet, sind hier nicht möglich.

XML-RPC - XML-Remote Procedure Call

In den späten 90ern entwickelt, kann XML-RPC als Vorgänger von SOAP angesehen werden. Während sich SOAP und andere Protokolle im Web Service Kontext weiter entwickelt haben, hat sich bei XML-RPC kaum etwas verändert. Die Stärke des XML-RPC Protokolls ist gleichermaßen seine Schwäche. Es spezifiziert lediglich einfache Punkt zu Punkt Verbindungen, die einen Prozeduraufruf und eine dazugehörige Antwort haben. Die Kommunikation der beteiligten Systeme ist bei XML-RPC nur über HTTP-POST Requests möglich. Namespaces, Routinginformationen oder die Wahl verschiedener Transportprotokolle sind alles Beispiele für Komplexität, die in XML-RPC fehlt. Damit bietet sich XML-RPC für sehr einfache Anwendungen an, die mit geringem Overhead und Implementationsaufwand auskommen. Nicht umsonst haben sich aber komplexere Spezifikationen immer weiter verbreitet, die den wachsenden Anforderungen der Maschine-zu-Maschine Kommunikation genügen können. Das nachfolgende Beispiel eines Funktionsaufrufes in XML-RPC sei angeführt, um die geringere Ausdrucksstärke und einfachere Syntax im Vergleich zu aktuelleren Protokollen zu verdeutlichen (vgl. [LAURENT et al. 2001], S. 20).

```
1 <?xml version="1.0" ?>
  <methodCall>
3    <methodName>print</methodName>
     <params>
5       <param>
           <value><string>Hello, World"</string></value>
7     </param>
     </params>
9 </methodCall>
```

Listing 2.1: XML-RPC Beispiellisting

J2EE

Innerhalb der J2EE-Spezifiktaion gibt es verschiedene Konzepte, die die Forderungen einer SOA erfüllen und so ihren Aufbau unterstützen können. So sind beispielsweise mit den Java Enterprise Beans Komponenten verfügbar, mit deren Hilfe höher granulierte Geschäftsprozesse abgebildet werden können. Jedoch finden sich in J2EE keine Spezifikationen für die Zusammenstellung solcher Dienste oder die nötige Schnittstellenbeschreibung. Erst weitere Frameworks schließen diese Lücken. Die Auffindbarkeit von Diensten wiederum lässt sich mit Java Technologien wie JNDI (Java Naming and Directory Interface) realisieren.

Web Services

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other system interact with the Web*

> *service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*
>
> vgl. W3C Working Group Note „Web Services Architecture“ (11.02.2004)

Web Services wurden erstmals im November 2002 im working draft „Web Services Architecture“ erwähnt und hiermit eine Sammlung von Spezifikationen zur Verfügung gestellt, die den Aufbau einer SOA unterstützen können. Das derzeit größte Missverständnis im Zusammenhang mit SOA und Web Services ist allerdings, dass der Einsatz von Web Services alleine bereits genüge, um eine SOA zu implementieren. Weitere von einer SOA geforderte Prinzipien sind dadurch aber noch nicht von sich aus erfüllt. Erst die Konzeption der Dienste anhand der in Kapitel 2.4.5 (siehe S. 10) vorgestellten Prinzipien und die damit einhergehende Ausrichtung an Geschäftsprozessen führen zu einer SOA. Web Services bedienen sich etablierter Standards aus dem Web-Umfeld und können damit leicht bestehende Infrastrukturen nutzen, sowie sich auf die Stabilität dieser Standards berufen. Da sich diese Diplomarbeit hauptsächlich der Einführung einer SOA mit Hilfe von Web Services widmet, ist eine ausführliche Betrachtung der relevanten Technologien, die in Zusammenhang mit Web Services Verwendung finden, im nachfolgenden Kapitel 2.5 (siehe S. 15zu finden.

2.5 Web Service Technologien und Erweiterungen

2.5.1 XML

XML - Extensible Markup Language - ist ein Standard des W3C, der die strukturierte Speicherung von Daten im Textformat ermöglicht. Bereits in den 70er Jahren unter der Bezeichnung Standard Generalized Markup Language begannen Charles Goldfarb, Ed Mosher und Ray Lorie bei IBM mit der Entwicklung dieser Sprache, die 1986 unter dem Namen XML als ISO-Standard 8879 angenommen wurde (Siehe [HAROLD et al. 2001], S. 8). Als Meta-Auszeichnungssprache definiert sie keine feste Menge von Elementen und ermöglicht es so, in allen Anwendungsgebieten flexibel eingesetzt zu werden. Diese Flexibilität steht einer strikten Grammatik gegenüber, die die Syntax eines XML-Dokuments vorgibt. Diesen Vorschriften genügende Dokumente werden als wohlgeformt bezeichnet. Zusätzliche Einschränkungen der Syntax einer XML-Datei können innerhalb einer DTD (Document Type Definition) definiert werden. In ihr wird unter anderem festgelegt, welche tags im Dokument und an welcher Stelle sie verwendet werden dürfen. Dokumente, die den Anforderungen einer ihr zugeordneten DTD entsprechen, werden als gültig bezeichnet. Im XML-Beispiellisting ist erkennbar, wie Meta-Auszeichnungen (Bezeichner in spitzen Klammern) den ausgezeichneten Text umschließen. Der Bezeichner person enthält außerdem ein Attribut (id="007"), um den ausgezeichneten Text weiter zu spezifizieren.

```
1 <?xml version="1.0" ?>
  <person id="007">
3    <name>Hans Mustermann</name>
     <telefon>
5       <mobil>+49 175 5692834</mobil>
        <privat>+49 40 5679986</privat>
7    </telefon>
     <adresse>
9       <strasse>Beispielstr. 101</strasse>
```

```
        <plz>21673</plz>
11      <ort>Hamburg</ort>
    </adresse>
13 </person>
```

Listing 2.2: XML-Beispiellisting

2.5.2 SOAP

Vormals wurde SOAP als Akronym für das 1998 eingeführte Protokoll „Simple Object Access Protocol“ verwendet. Seit Version 1.2 ist SOAP der eigenständige Name des Protokolls. Ursprünglich als Protokoll eingeführt, das den Zugriff auf Objekte ermöglichen sollte (daher der Name „Simple Object Access Protocol“), entwickelte es sich bis zur heutigen Version zu einem Protokoll, das dem Austausch von strukturierten Nachrichten in einer dezentralen und verteilten Umgebung dient und damit wesentlich flexibler ist. Die Namensänderung wurde mit Version 1.2 vorgenommen, um den möglicherweise verwirrenden Fokus vom Objektzugriff zu nehmen. Als Protokoll für den Nachrichtenaustausch kommt SOAP bei Web Services zum Einsatz.

SOAP Knoten

SOAP Knoten können die nachstehend beschriebenen Rollen übernehmen. Welche Rolle ein Knoten bei der Verarbeitung einer Nachricht einnimmt, entscheidet darüber, wie und welche Header der Nachricht verarbeitet werden. Ausgewählte SOAP Header können mit Hilfe des „actor“-Attributs für bestimmte Rollen als verbindlich markiert werden. Kann ein entsprechender Knoten diese Header nicht verarbeiten, so muss ein SOAP-Fehler generiert und die Verarbeitung der Nachricht abgebrochen werden. Nicht für eine bestimmte Rolle markierte Header können von entsprechenden Knoten ignoriert werden. Neben der in SOAP 1.1 definierten Rolle „http://schemas.xmlsoap.org/soap/actor/next/“ können eigene Rollen definiert werden, um eine entsprechende Anwendungslogik abzubilden.

Abbildung 2.2: SOAP-Knoten in einer mehrstufigen Übertragung

In Abbildung 2.2 sind die drei Arten von Knoten erkennbar:

SOAP-Absender Absender und in der Regel auch Erzeuger einer Nachricht.

SOAP-Empfänger Der letzte Empfänger einer Nachricht verarbeitet neben den Headern der Nachricht in der Regel auch den kompletten Inhalt der Nachricht, die an ihn adressiert wurde.

SOAP-Zwischenstelle Eine SOAP-Zwischenstelle ist ein Knoten, der sowohl als Empfänger, als auch als Sender agiert. Der Nachrichteninhalt wird von Zwischenstellen in den meisten Fällen ignoriert.

SOAP Binding Framework

SOAP Nachrichten können über eine Vielzahl von Transportprotokollen ausgetauscht werden. Die Vorschriften, wie eine Transportschicht eine SOAP-Nachricht zu behandeln hat, sind innerhalb von protokollspezifischen bindings definiert. Das „SOAP Protocol Binding Framework“ gibt dabei vor, wie ein solches binding spezifiziert werden muss. Innerhalb der W3C Recommendation für SOAP ist lediglich das HTTP-Binding definiert, es existieren aber eine Vielzahl weiterer Spezifikationen für andere Transportprotokolle.

SOAP Beispielnachricht

Die nachfolgende Beispiel SOAP-Nachricht stammt aus der Recommendation für SOAP vom W3C (Siehe [GUDGIN et al. 2007a]). In diesem einfachen Beispiel ist der Aufbau einer SOAP-Nachricht bereits erkennbar. Ob Informationen im Header oder Body stehen, hängt in der Regel davon ab, wem diese Informationen nutzen. Daten im Header sind nicht nur für den endgültigen Empfänger der Nachricht von Interesse, sondern werden ggfs. auch von Stationen berücksichtigt, die die Nachricht nur weiterleiten. So könnte eine Übermittlungsstation die Nachricht aus Beispiel 1 bevorzugen, da in ihrem Header eine hohe Priorität vergeben wurde. Der eigentliche Dateninhalt im Body, also der Text der Alarmierung selbst, ist für eine Station, die die Nachricht nur weiterleitet, hingegen nicht von Interesse. Durch diese Unterteilung der Informationen lässt sich die Verarbeitungsgeschwindigkeit von Nachrichten auf den Übertragungsstationen erheblich verbessern.

```
1  <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
     <env:Header>
3      <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
         <n:priority>1</n:priority>
5        <n:expires>2001-06-22T14:00:00-05:00</n:expires>
       </n:alertcontrol>
7    </env:Header>
     <env:Body>
9      <m:alert xmlns:m="http://example.org/alert">
         <m:msg>Pick up Mary at school at 2pm</m:msg>
11     </m:alert>
     </env:Body>
13 </env:Envelope>
```

Listing 2.3: Beispiel 1: SOAP-Nachricht (Siehe [GUDGIN et al. 2007a])

SOAP Message Exchange Pattern (MEP)

Message Exchange Pattern (MEP) definieren, wann und wie Nachrichten von Kommunikationspartnern ausgetauscht werden. Dabei hängt es vom MEP ab, welcher Form die Nachrichten entsprechen müssen, die ausgetauscht werden und ob Nachrichten erforderlich oder optional sind. Inbound und Outbound Nachrichten, die in MEP ausgetauscht werden, sind nicht näher spezifiziert. Sind in den MEP SOAP-Nachrichten gefordert, so müssen sie von

den Knoten, die diese Nachrichten senden, empfangen oder weiterleiten, gemäß dem SOAP Processing Model (vgl. [GUDGIN et al. 2007a], Kapitel 2) verarbeitet werden. Typen von Sender und Empfänger in den Nachrichten hängen vom Transport-Binding ab und sind beim HTTP-Transport in der Regel vom Typ xs:anyURI. Empfangende Knoten, unabhängig davon, ob sie die Nachricht weiterleiten oder der endgültige Zielknoten sind, müssen bei einer erfolgreich empfangenen Nachricht das Attribut InboundMessage mit dem Inhalt des Attributs OutboundMessage füllen.

SOAP One-way Message Exchange Pattern

Dieses Message Exchange Pattern definiert die Übertragung von keiner oder mehr SOAP-Nachrichten von genau einem sendenden SOAP-Knoten zu keinem oder mehreren empfangenden SOAP-Knoten (vgl. [ORCHARD 2007], Kapitel 2). Es eignet sich typischerweise zum Versenden von Benachrichtigungen.

SOAP Request-Response Message Exchange Pattern

In diesem MEP sind exakt zwei Nachrichten vorgesehen. Die Request-Nachricht muss eine SOAP-Nachricht sein. Die Antwort kann entweder eine SOAP-Nachricht oder eine vom Transport-Binding abhängige Empfangsbestätigung sein. (vgl. [GUDGIN et al. 2007b], Kapitel 6.2)

SOAP Response Message Exchange Pattern

Wie im Request-Response MEP wird auch hier eine Nachricht für eine Anfrage gesendet. Diese Nachricht ist jedoch im Gegensatz zum Request-Response MEP keine SOAP-Nachricht. Die Antwort hingegen muss eine SOAP-Nachricht sein und von den Knoten, die sie senden bzw. empfangen, entsprechend verarbeitet werden. Dieses MEP spezifiziert keine Zusammenhänge zwischen mehreren Anfragen und ist beschränkt auf den Austausch der Antwortnachricht. Als eine Vereinfachung des Request-Response MEP lässt sich dieses MEP verwenden, sofern für eine Anfrage keine Informationen erforderlich sind und eine Antwort lediglich angestoßen werden muss.

2.5.3 WSDL

Die WSDL ist ein XML-Format, das die Rahmenbedingungen für die Beschreibung von Endpunkten, Nachrichten und Datentypen definiert, die in einem Web Service Verwendung finden.
Anhand des Beispiels aus [CERAMI 2002] (Kapitel 6, „The WSDL Specification“, S. 120ff) werden nachfolgend die Abschnitte eines WSDL-Dokuments erläutert.

definitions

Der Abschnitt „definitions“ enthält im Wesentlichen alle Namensräume, die in der WSDL zur Verwendung kommen. Diese können in späteren Elementen dann mit ihren Abkürzungen referenziert werden. Darüber hinaus wird ein Standardnamensraum definiert, der von allen Elementen verwendet wird, bei denen kein Namensraum angegeben ist (hier: xmlns=„http://schemas.xmlsoap.org/wsdl/“).

```
<definitions name="HelloService"
   targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   ...
</definitions>
```

Listing 2.4: Beispiel WSDL: definitions

message

Der Abschnitt „message“ kann beliebig häufig auftreten und definiert sämtliche Nachrichten, die von diesem Web Service empfangen oder gesendet werden. Pro Nachricht werden alle enthaltenen Attribute mit ihrem Datentyp definiert.

```
<message name="SayHelloRequest">
   <part name="firstName" type="xsd:string"/>
</message>
<message name="SayHelloResponse">
   <part name="greeting" type="xsd:string"/>
</message>
```

Listing 2.5: Beispiel WSDL: message

port type

Im „portType“-Abschnitt werden alle Operationen definiert, die vom Web Service unterstützt werden. Im vorliegenden Beispiel ist dies lediglich die Operation „sayHello“.

```
<portType name="Hello_PortType">
   <operation name="sayHello">
      <input message="tns:SayHelloRequest"/>
      <output message="tns:SayHelloResponse"/>
   </operation>
</portType>
```

Listing 2.6: Beispiel WSDL: portType

In der WSDL-Spezifikation 1.1 werden 4 verschiedene Typen von Operationen unterstützt, bei denen die jeweils verwendeten Nachrichten zu definieren sind.

- One-way: Der Web Service empfängt lediglich eine Nachricht. Die Operationsdefinition enthält nur ein „input“-Element.
- Request-response: Der Web Service empfängt eine Nachricht und versendet anschließend eine Antwort. Die Operationsdefinition enthält ein „input“-Element gefolgt von einem „output“-Element.
- Solicit-response: Der Web Service sendet eine Nachricht und erwartet anschließend eine Antwort. Die Operationsdefinition enthält ein „output“-Element gefolgt von einem „input“-Element.
- Notification: Der Web Service versendet eine Nachricht. Die Operationsdefinition enthält nur ein „output“-Element.

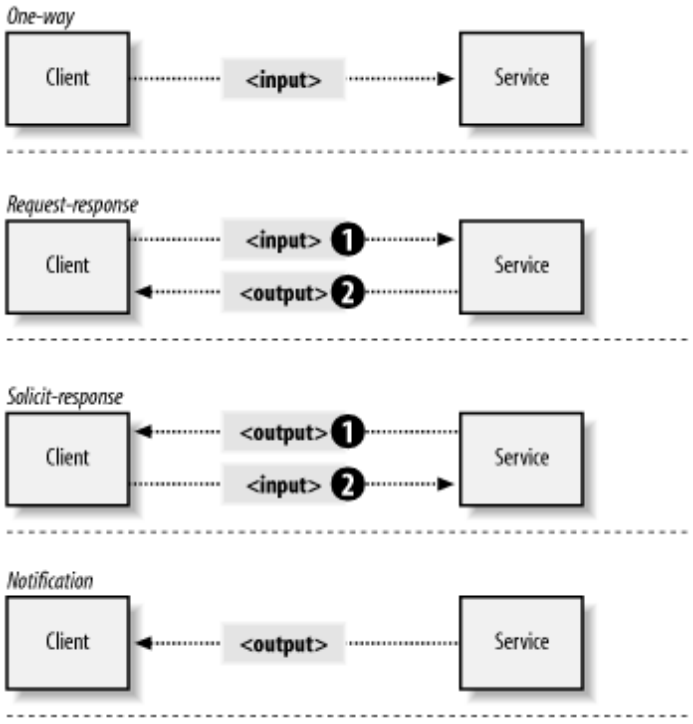

Abbildung 2.3: Operationstypen die in WSDL 1.1 unterstützt werden, [CERAMI 2002]

binding

Der „binding“-Abschnitt spezifiziert, wie die Nachrichten der von Web Service zur Verfügung gestellten Operationen übertragen werden müssen. Im nachfolgenden Beispiel ist die vom Web Service angebotene Operation per http verfügbar. Für die ausgetauschten Nachrichten wird das SOAP-encoding gewählt sowie der verwendete Namensraum angegeben. Spezifischere Informationen zu den einzelnen Elementen finden sich in [CERAMI 2002] (Kapitel 6, „The WSDL Specification“, S. 120ff).

```
<binding name="Hello_Binding" type="tns:Hello_PortType">
    <soap:binding style="rpc"
       transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHello">
       <soap:operation soapAction="sayHello"/>
       <input>
          <soap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="urn:examples:helloservice"
             use="encoded"/>
       </input>
       <output>
          <soap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="urn:examples:helloservice"
             use="encoded"/>
       </output>
    </operation>
 </binding>
```

Listing 2.7: Beispiel WSDL: binding

service

Der „service“-Abschnitt enthält für jeden binding-Element die Definition eines ports. Diese Definition verbindet das zugehörige binding mit der Adresse, unter der der port erreichbar ist. Im Falle des vorliegenden Beispiels ist dies die URL des entsprechenden servlets.

```
<service name="Hello_Service">
    <documentation>WSDL File for HelloService</documentation>
    <port binding="tns:Hello_Binding" name="Hello_Port">
       <soap:address
          location="http://localhost:8080/soap/servlet/rpcrouter"/>
    </port>
  </service>
```

Listing 2.8: Beispiel WSDL: service

2.5.4 WS-Security

Die Erweiterung WS-Security[3] stellt mit dem Hauptbestandteil der Spezifikation „Web Services Security: SOAP Message Security 1.1“ drei Mechanismen zur Verfügung, um Geheimhaltung und Integrität von Nachrichten sowie Authentifizierung zu ermöglichen. Die Spezifikation unterstützt den Einsatz verschiedener Mechanismen wie beispielsweise PKI, Kerberos oder SSL. Eine vollständige Liste der unterstützen Formate und Technologien ist in den jeweiligen Kapiteln der Spezifikation zu finden.

Sicherheitsmerkmale in Nachrichten

Der für das CommSy besonders interessante Teil der Spezifikation ermöglicht es, Authentifizierungen abzuwickeln. In der Spezifikation sind verschiedene Typen von sogenannten SecurityTokens vorgesehen. Der einfachste und im Rahmen des CommSy einsetzbare Typ ist die gängige „Benutzername-Passwort“ Kombination. Im nachstehenden Listing sind die im wsse:Security Element anzugebenden tags ersichtlich, die diese Form der Authentifizierung ermöglichen. Weitere Möglichkeiten zur Authentifizierung wären beispielsweise die Verwendung von X.509 Zertifikaten und Kerberos Tickets.

wsse:Username Dieser tag enthält den Benutzernamen.

wsse:Password Das Passwort des Benutzers kann entweder im Klartext oder als Hash übertragen werden. Der Typ des tags endet dann mit #PasswordDigest oder #PasswordText.

```
<wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
    wss-username-token-profile-1.0#PasswordText">password</wsse:Password>
```

Listing 2.9: PasswordType: Klartext

Wird das Passwort mit dem Typ PasswordDigest übertragen, so wird dieser aus der Kombination Passwort, wsse:Nonce und wsse:Created berechnet:

```
PasswordDigest = Base64(SHA-1(Nonce + Created + Password))
```

[3] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

wsse:Nonce Ein zufälliger Wert, der sich bei jeder Anfrage ändern sollte, um Replay-Angriffe zu erschweren.

wsse:Created Dieser optionale Zeitstempel gibt in UTC an, wann das Authentifizierungsmerkmal erzeugt wurde.

```
<soapenv:Header>
   <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.
       oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext
       -1.0.xsd">
      <wsse:UsernameToken wsu:Id="UsernameToken-6672618" xmlns:wsu="
          http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
          wssecurity-utility-1.0.xsd">
         <wsse:Username>joe</wsse:Username>
         <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/
             oasis-200401-wss-username-token-profile-1.0#PasswordDigest
             ">fEYQ0mJHfqKENsNeVYfhP50msFE=</wsse:Password>
         <wsse:Nonce>OU/rR5LXBoLx5udc/UzifQ==</wsse:Nonce>
         <wsu:Created>2008-11-18T14:54:52.029Z</wsu:Created>
      </wsse:UsernameToken>
   </wsse:Security>
</soapenv:Header>
```

Listing 2.10: Beispiel WS-Security Header

Integrität der Nachrichten

Um die Integrität der Nachrichten zu gewährleisten, lassen sich mit Hilfe der WS-Security Spezifikation diese oder nur Teile der Nachricht signieren. Dass diese Aufgabe verschiedene Schwierigkeiten birgt, zeigt die Diskussion in Kapitel 8.1 „Algorithms“ der Spezifikation. Da es erlaubt ist, XML-Dokumente auf verschiedene Arten zu manipulieren und dabei äquivalente Dokumente zu erzeugen, ist es möglich und wahrscheinlich, dass Nachrichten auf ihrem Weg durch verschiedene Knoten vom Sender bis zum Empfänger nicht identisch sind. Eine Signatur, die lediglich auf Basis des gesamten Dokumentes erstellt wird und Äquivalenz erhaltende Manipulationen von XML-Dokumenten ignoriert, kann also nicht ausreichen. Als einfaches Beispiel sei an dieser Stelle das Löschen doppelt definierter Namensräume oder Expandieren von Namensräumen genannt. Eine ausführliche Diskussion findet sich im oben genannten Kapitel. Es stehen zwei verschiedene Algorithmen zur Verfügung, die das Signieren der Nachrichteninhalte ermöglichen und versuchen, diese Konflikte zu lösen.

Vertraulichkeit der Nachrichten

Kapitel 9 „Encryption“ der Spezifikation erläutert die Möglichkeiten, Body oder Header der SOAP-Nachrichten zu verschlüsseln. Dabei werden die bereits durch den XML Encryption Standard[4] etablierten Mechanismen eingesetzt. Der zur Verschlüsselung verwendete symmetrische Schlüssel kann entweder bei Sender und Empfänger als bekannt vorausgesetzt werden oder mit einem öffentlichen Schlüssel des Empfängers verschlüsselt im

[4] http://www.w3.org/Encryption/2001/

„xenc:EncryptedKey“-Element als Teil der Nachricht enthalten sein. Da eine Verschlüsselung der Nachrichten für die hier umgesetzten Web Services nicht geplant ist, soll dies als Ausblick auf die Möglichkeit der Verschlüsselung genügen.

2.5.5 WS-Notification

Mit WS-Notification[5] steht eine Sammlung von Spezifikationen zur Verfügung, die ereignisbasierte Benachrichtigungen mit Hilfe von Web Services ermöglichen sollen. Es ist damit möglich, PUSH-Szenarien zu unterstützen. Eine wichtige Eigenschaft, die diese Erweiterung sehr mächtig macht, ist die Tatsache, dass ein Web Service, der das Abonnieren von Benachrichtigungen anbietet, erst zur Laufzeit Kenntnis über die Abonnenten erlangen muss.

[5] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

Kapitel 3

3 Mehrwerte durch SOA

Mehrwerte einer SOA ergeben sich an vielen Stellen. Daher lohnt es sich, die Untersuchung der Mehrwerte, die durch die Implementierung einer SOA erreicht werden können, in die verschiedenen Aspekte zu gliedern, die einer SOA zu Grunde liegen.

3.1 Architektonische Mehrwerte

Um eine dienstorientierte Architektur erfolgreich in einem Softwaresystem umzusetzen, ist eine saubere Trennung der Funktionalität von ihrer Präsentation notwendig. Die Dienste einer solchen Architektur stellen einen weiteren Einstiegspunkt in das Softwaresystem dar und greifen gemeinsam mit der Präsentationsschicht auf die gleichen Methoden der Funktionsschicht zu. Ist die Schnittstelle der Dienste sauber definiert und lässt sich so in zukünftigen Iterationen des Softwaresystems abwärtskompatibel weiterentwickeln, stellt sie eine wichtige Komponente des Systems dar, wenn es um die Realisierung des Zugriffs unterschiedlichster Clients auf das System geht. Nachfolgend sind daher einige Vorteile näher erläutert, die sich aus den Eigenschaften von Diensten in einer SOA ergeben.

3.1.1 Mehrwerte durch Zustandslosigkeit

Die Forderung, dass Dienste einer SOA zustandslos implementiert werden, ist oftmals schwer umsetzbar und die sich daraus ergebenden Mehrwerte sind nicht sofort ersichtlich. Langfristig erhält und unterstützt die Zustandslosigkeit aber die weiteren Mehrwerte, die eine SOA zu bieten hat. Die bisherige Erfahrung zeigt, dass sich die flexible Kombinierbarkeit bei zustandslosen Diensten einfacher und länger aufrecht erhalten lässt. Darüber hinaus ist durch die Zustandslosigkeit eine geringere Ressourcennutzung auf dem Server zu erwarten. Im Gegensatz zu zustandsbehafteten Diensten müssen die benötigten Daten hier nur für die Dauer des Requests und dessen Bearbeitung vorgehalten werden. Eine Zuordnung von gespeicherten Zustandsinformationen zu späteren Aufrufen entfällt. Im Einzelfall kann dieser Vorteil aber durch einen aufwändigen Initialisierungsprozess aufgewogen werden. Unter Umständen ist abzuwägen, ob ein Authentifizierungsprozess oder die Erzeugung aller zur Bearbeitung des Requests benötigter Daten so aufwändig ist, dass ein Vorhalten der Daten in einer Session besser wäre.

3.1.2 Mehrwerte durch Plattformunabhängikeit

Lange Zeit war es von Herstellern umfangreicher Softwaresysteme nicht erwünscht, Interoperabilität mit anderen Systemen zu ermöglichen. Erst der erhöhte Druck von Unternehmen, die immer häufiger vor Integrationsproblemen in B2B-Projekten standen, ebnete den Weg für Bestrebungen diese Interoperabilität zu ermöglichen. Mit plattformunabhängigen Protokollen und der Fähigkeit Daten zwischen Systemen auszutauschen, deren Implementierung nicht relevant für den Datenaustausch selbst ist, bieten SOAs die ideale Grundlage für solche Integrationen. Softwaresysteme lassen sich dadurch leichter und weniger fehleranfällig zusammenbringen. IT-Systeme können somit wesentlich reibungsloser an die sich wandelnden Anforderungen einer durch sie unterstützten Geschäftswelt angepasst werden.

3.1.3 Mehrwerte durch Multi-Client-Access

Durch die Plattformunabhängigkeit ergeben sich direkt die Vorteile einer SOA für die Unterstützung verschiedener Client-Systeme. Die in einer SOA zur Verfügung gestellten Dienste sind frei von Wissen über Systeme implementiert, die sie später verwendenden werden. Daher ist es bei gut konzipierten Diensten einer SOA sehr einfach möglich, neue Client-Systeme einzubinden. Die Dienste selbst bleiben davon unberührt.

3.1.4 Mehrwerte durch Dienst-Orchestrierung

Die Funktionalität, die durch einzelne Dienste angeboten wird, ist in der Regel eine kleinstmögliche Funktionseinheit. Oft erzielt erst das Zusammenspiel mehrerer Dienste in der durch einen Geschäftsprozess vorgegebenen Kombination und Reihenfolge ein entsprechendes Ergebnis. So orchestrierte Dienste können wiederum als Dienst zur Verfügung gestellt werden, um einen einfachen und schnellen Zugriff auf den gesamten Prozess zu ermöglichen. Die Kombinierbarkeit der einzelnen Dienste erlaubt eine flexible und schnelle Reaktion auf Änderungen in Prozessabläufen. Ändern sich Prozesse, ohne dass sich die verwendeten Daten ändern, können im Idealfall Schnittstellen kompatibel gehalten werden und die Anpassung der Dienste bleibt für die Clients unbemerkt. Auch wenn Änderungen nur die Abläufe einzelner Clients betreffen, können diese ohne serverseitige Anpassungen leicht durch die Clients umgesetzt werden.

3.2 Mehrwerte für den Benutzer

Der treibende Faktor für Implementierungen in Unternehmen ist in der Regel zunächst einmal zu erwartende Kostenersparnis. Durch die Implementierung einer SOA können sich Kostenersparnisse in verschiedenen Lebenszyklen eines Softwaresystems ergeben. In den meisten Fällen ergibt sich aus unternehmerischer Perspektive ein spürbarer Mehrwert allerdings erst nach einer gewissen Zeit. Ist eine SOA gut konzipiert und implementiert, so lassen sich neue oder veränderte Prozessanforderungen auf Geschäftsebene in der Regel schneller umsetzen. Die dadurch entstehende Kostenminimierung und der Wettbewerbsvorteil auf Grund der Zeitersparnis sind Mehrwerte, die sich nicht in der Implementierungsphase einer SOA erzielen lassen. Eine stärkere Verknüpfung von IT mit den Geschäftsprozessen vereinfacht den Dialog zwischen Entwicklern und Benutzern deutlich. Ist auf der Implementierungsseite das Bewusstsein für die Prozesse der Unternehmenswelt geschärft und

Grundlage für die Konzeption, so lassen sich Anforderungen der Benutzer leichter verstehen und umsetzen.

3.3 Technische Mehrwerte

Werden die in einer SOA geforderten Prinzipien umgesetzt, so ergeben sich auch während der Implementierungsphase verschiedene Mehrwerte. Aus den erzielbaren Vorteilen ergibt sich in der Regel eine schnellere und damit kostengünstigere Entwicklung. Mittel- bzw. langfristig führt in der Regel geringerer Wartungsaufwand zu weiterer Optimierung der Entwicklung (vgl. [NEWCOMER und LOMOW 2005]).

3.3.1 Effizientere Softwareentwicklung

Die in einer SOA geforderte Modularität der Dienste wirkt sich positiv auf deren Entwicklung aus. Die einzelnen Dienste können, nachdem ihre Schnittstellen definiert sind, unabhängig voneinander implementiert werden. Darüber hinaus ist es möglich, Server und Client Komponenten parallel auf Basis der Schnittstellenbeschreibungen zu entwickeln. Die Möglichkeit, Dienste unabhängig von einander zu implementieren, ermöglicht die schrittweise Einführung umfangreicher Systeme, in denen bereits nach Umsetzung weniger zentraler Dienste die Basisfunktionalität genutzt werden kann. In einzelnen Iterationen können dann Dienste hinzugefügt werden, die weitere Funktionalität bieten, ohne die bisher eingeführten Dienste zu beeinträchtigen.

3.3.2 Wiederverwendbarkeit

Die verschiedene Charakteristika, die einer SOA inhärent sind, führen zu einer hohen Wiederverwendbarkeit der Dienste. Konsequente Modularität und vor allem Plattformunabhängigkeit tragen erheblich zu gesteigerter Wiederverwendbarkeit der Dienste bei.

3.4 Zusammenfassung

Die Untersuchungen in diesem Kapitel zeigen, dass die Vorteile der durch eine SOA zur Verfügung gestellten Dienste in allen Bereichen im Wesentlichen durch Flexibilität und Transparenz entstehen. Sowohl für die Entwickler, als auch für die Benutzer ergibt sich durch die erhöhte Relevanz der Kommunikation zur Ausrichtung der SOA-Dienste an Geschäftsprozessen eine bessere Wahrnehmung für die Beschaffenheit und Anforderungen der jeweils anderen Seite. Darüber hinaus zeigen sich auch einige Vorteile, die ausschließlich der Entwicklung des Softwaresystems zugutekommen.

Kapitel 4

4 Planung der Implementierung

Neben den funktionalen Anforderungen, die die Web Services zu erfüllen haben, beschäftigt sich die Arbeit mit den architektonischen Anforderungen, die erfüllt sein müssen, um effizienter Bestandteil einer SOA zu sein. Die Untersuchungen sind unterteilt in die Anforderungen an die Dienste selbst (vgl. Kapitel 2.4.5, S. 10) und die Anforderungen, die die Architektur des Softwaresystems betreffen, in das die Dienste integriert werden. Die Anforderungen an die Dienste wurden bereits bei deren Entwurf berücksichtigt. Den Anforderungen an das Softwaresystem widmet sich Kapitel 5.1 (siehe S. 35ff).

4.1 Auswahl von Aspekten einer SOA für das CommSy

Die Einführung der Begriffe und Konzepte einer SOA im Grundlagenkapitel zeigt auf, dass die Implementierung einer SOA ein sehr umfangreiches Projekt ist. Die im Rahmen dieser Diplomarbeit umsetzbaren und vor allem im Kontext des CommSy sinnvollen Aspekte beschränken sich auf die zentralen Konzepte einer SOA. Orchestrierung von Diensten mit Hilfe von BPEL oder das Einrichten eines Verzeichnisses für angebotene Dienste oder Datentypen erschien für das CommSy unnötig. Erst wenn weitere Dienstanbieter mit in eine SOA aufgenommen werden oder das CommSy Teil einer bestehenden SOA werden soll, können solche Aspekte ausreichend untersucht werden. Der Fokus der Arbeit ist auf die zentrale Komponente einer SOA, den Dienst, gerichtet. Durch die Implementierung des Prozesses „Lesung in Österreich“ werden wesentliche Eigenschaften von Diensten im Zusammenspiel mit anderen Diensten untersucht.

4.2 Warum Web Services für das CommSy?

Welche Technologien für den Aufbau einer SOA zum Einsatz kommen ist stark vom Einsatzgebiet abhängig und sollte anhand verschiedener Kriterien beurteilt werden. Web Services sind, was ihre Performance angeht, den meisten anderen Implementierungen unterlegen. Binäre Protokolle wie beispielsweise JAVA-RMI oder CORBA sind deutlich schneller. Im Vergleich zum eigentlich Nutzinhalt erzeugt der Overhead, der bei Web Services durch den Einsatz von XML entsteht, ein um ein vielfach größeres Datenvolumen. Auch das aufwendige De-/Serialisieren der ausgetauschten Objekte führt zu Performance-Einbußen auf

Server- und Clientseite. Im Vergleich zu binären Protokollen wird hier mehr Rechenzeit benötigt, bevor der eigentlich Nutzinhalt der Nachrichten in einer Form (meist XML) zur Verfügung steht, die von den beteiligten Systemen verarbeitet werden kann. Dem gegenüber stehen aber Vorteile durch größtmögliche Plattformunabhängigkeit. Im Kontext des CommSy ist nicht mit extremen Anforderungen an Skalierbarkeit oder Latenzzeiten zu rechnen. Die erwähnten Geschwindigkeitsvorteile bzw. die Entlastung der beteiligten Systeme, die durch die Verwendung von binären Protokollen zu erwarten sind, haben für die Wahl der Technologie demnach nur geringe Relevanz. Plattformunabhängigkeit hingegen erschien deutlich wichtiger. Der Einsatz von Web Services und die Vielzahl der zur Verfügung stehenden Frameworks stellen sicher, dass der Integrationsaufwand so klein wie möglich gehalten werden kann. Hierbei ist es egal, ob von einem BlackBerry Client, einem Widget oder möglicherweise einem Rich-Client aus mit den Web Services gearbeitet werden soll.

4.3 Abgrenzung der implementierten Funktionalität

Für die im Rahmen dieser Diplomarbeit implementierten Web Services musste entsprechend der zur Verfügung stehenden Zeit eine Auswahl der umgesetzten Funktionalität getroffen werden. Entgegen der ersten Annahme, dass Web Services nur für Kategorieren implementiert werden können, die bereits komplett im CommSy umgesetzt sind, hat sich gezeigt, dass die benötigten Services bereits für alle Kategorien zur Verfügung stehen. Eine horizontale Abgrenzung der umzusetzenden Funktionalität findet daher nicht statt. Die zeitgleich entstehende Diplomarbeit „Auswirkungen auf eine bestehende ereignisgetriebene Architektur durch die Erweiterung um eine aktive Signalisierung“ von Matthias Hager wird die umgesetzten Web Services zur Implementierung eines BlackBerry-Clients nutzen, sie wird somit in die Anforderungsanalyse einfließen und die Auswahl der zu implementierenden Funktionalität beeinflussen.

4.3.1 Priorisierung der umzusetzenden Funktionalität

Um die Web Services sinnvoll einsetzen zu können, sind eine Reihe von Funktionen mindestens umzusetzen. Das Problem, dass durch die Erweiterung des CommSy um Web Services in erster Linie gelöst werden soll, ist die Notwendigkeit ständig in einzelnen oder sogar mehreren Portalen eingeloggt sein zu müssen, um sich zeitnah über Änderungen im CommSy zu informieren. Mit Hilfe der Web Services lassen sich Tools wie beispielsweise ein Desktopwidget oder der BlackBerry Client realisieren, die mit dem CommSy interagieren können. Für dieses Szenario ist zunächst der lesende Zugriff auf das CommSy nötig. Als wichtiger Bestandteil dieses lesenden Zugriffs gilt es, die im CommSy wirksamen Berechtigungen eines Benutzers zu berücksichtigen. Höchste Priorität bei der Umsetzung der Web Services erhielt demnach die Integration des WS-Security Frameworks zur sicheren Authentifizierung eines Benutzers bei der Benutzung der Web Services. Im Rahmen der Diplomarbeit von Matthias Hager ist es darüber hinaus nötig, dass ein Benutzer sich für Benachrichtigungen zu bestimmten Räumen an- bzw. abmelden kann. Um diese Funktionalität direkt aus dem Client verwenden zu können, wurden entsprechende Web Services mit umgesetzt.

4.3.2 Orchestrierung von Diensten

Um auch auf Eigenschaften einer SOA eingehen zu können, die nicht dienstspezifisch sind, wird ein beispielhafter Prozess umgesetzt, der sowohl auf eigene, als auch auf externe Dienste angewiesen sein wird. Anhand dieses Dienstes wird es möglich sein, die Charakteristika von Diensten zu diskutieren, die sich durch die Orchestrierung ändern. Für die Diskussion wird eine dreistufige Komplexitätssteigerung angenommen. Eigene, atomare Dienste werden im Kontext der Orchestrierung als Dienste der niedrigsten Komplexität angenommen. Wird ein Dienst aus mehreren selbstverwalteten bzw. implementierten Diensten orchestriert, so wird er der zweiten Komplexitätsstufe zugeordnet. Durch die Hinzunahme externer Dienste in die Orchestrierung wird zusätzliche Komplexität eingeführt, die zur dritten Stufe dieser Betrachtung führt. Anhand dieses Beispiels sollen Herausforderungen erläutert werden, denen man sich auf den unterschiedlichen Komplexitätsstufen zu stellen hat.

4.3.3 Ausblick auf nicht implementierte Funktionalität

Mit WS-Extensions stehen vielfältige Möglichkeiten zur Verfügung, die Funktionalität der Web Services zu erweitern. Darüber hinaus ist auch die Funktionalität, die das CommSy bietet, durch die bisherige Implementierung nur zum Teil abgedeckt. Nachfolgende Ausblicke auf mögliche Erweiterungen sollen einen Anreiz für die mögliche Weiterentwicklung der mit dieser Diplomarbeit gelegten Grundlage schaffen.

WS-Notification

Mit WS-Notification steht eine stabile API für ereignisbasierte Benachrichtigungen zur Verfügung. Es wird unter anderem ein publish/subscribe Szenario spezifiziert, das im Rahmen des CommSy für Benachrichtigungen bei Aktualisierungen in Räumen zum Einsatz kommen könnte. Der Web Service könnte verschiedene Benachrichtigungstypen anbieten, bei denen sich die Clients an- bzw. abmelden können. So könnte man sich bei neu eingestellten Dokumenten, neuen Nachrichten oder ähnlichen Ereignissen benachrichtigen lassen. Die Unterstützung von Informations-PUSH zum Client im Gegensatz zu regelmäßigem POLLING würde sich positiv auf viele Bereiche auswirken. Geringerer Netzwerktraffic würde für mobile Endgeräte zu längerer Batterielaufzeit und geringeren Kosten führen. Die deutlich geringere Anzahl der Service-Aufrufe auf dem Server reduziert die nötigen Hardware-Ressourcen. Geringere Latenz zwischen Ereignis und dessen Wahrnehmung durch den Benutzer erhöht die Akzeptanz der Lösung durch den Benutzer. Die für ereignisbasierte Benachrichtigungen nötigen Refactorings sind bereits im Rahmen der Diplomarbeit von Matthias Hager umgesetzt worden. Eine Einbindung dieser Benachrichtigungen in die WS-Notification API sollte mit überschaubarem Aufwand möglich sein.

WS-Security

Die Implementierung der WS-Security Erweiterung beschränkt sich auf die Authentifizierung. WS-Security bietet darüber hinaus weitere Möglichkeiten, die Sicherheit der Web Services zu verbessern. (vgl. Kapitel 2.5.4, S. 21) Da die Integration von WS-Security bereits erfolgreich umgesetzt ist, sollte eine Erweiterung um Signaturen oder Verschlüsselung

mit geringem Aufwand möglich sein. Von einer Implementierung im Rahmen dieser Diplomarbeit wurde abgesehen, da im Kontext des CommSy Vertraulichkeit und Authentizität keine ausreichende Relevanz zugemessen wurde. Die Umsetzung der Verschlüsselung würde des Weiteren den Implementierungsaufwand für Clients, denen kein Framework mit WS-Security Unterstützung zur Verfügung steht, erheblich erhöhen. Die durch Matthias Hager umgesetzte Implementierung des BlackBerry-Clients wäre dadurch im Rahmen seiner Diplomarbeit nicht zu bewerkstelligen gewesen.

Umsetzung weiterer Funktionalität

Die bisher umgesetzten Web Services beschränken sich weitestgehend auf den lesenden und schreibenden Zugriff auf die Basistypen des CommSy, um Clients die Möglichkeit zu geben, immer auf dem aktuellen Stand zu sein. Mit Hilfe weiterer Web Services, die beispielsweise auch das Hochladen von Materialen ermöglichen, wäre es möglich, Clients mit vollem Funktionsumfang auf Basis der Web Services zu erstellen, die die Weboberfläche als primäres Benutzerinterface ablösen könnten.

Einbindung externer Dienste

Die beiden zur Orchestrierung des Prozesses „Lesung in Österreich“ eingebundenen Dienste stellen die verwendete Funktionalität nur rudimentär zur Verfügung. Sowohl die Überprüfung einer Adresse als auch die Plausibilitätsprüfung der Buch-Daten ließe sich umfangreicher und komfortabler umsetzen. Die verwendete Adressprüfung stellt lediglich sicher, dass die übergebene Postleitzahl existiert und zu dem angegebenen Ort in Österreich gehört. Stünde ein Dienst zur Verfügung, der auf Basis eines größeren Datenbestandes arbeitet, so könnte beispielsweise auch die Existenz einer Straße sicher gestellt werden, die diesem als Parameter übergeben wurde. Auch wäre es besser, diese Überprüfungen nicht nur für Österreich zu ermöglichen. Der zur Plausibilitätsprüfung der ISBN verwendete Web Service prüft lediglich, ob das Format und die Prüfsumme der ISBN korrekt sind. Denkbar wäre die Verwendung eines Web Services, der auf Grund der übermittelten ISBN weitere Informationen zu dem entsprechenden Buch zurückliefert. Leider standen für beide Szenarien zum Zeitpunkt der Diplomarbeit keine kostenlosen Dienste zur Verfügung. Für die Abfrage der Buchdaten ließe sich alternativ auf die von Amazon angebotenen Web Services[1] zurückgreifen. Diese setzen allerdings eine Registrierung und einen dazu gehörenden Developer-Key voraus.

4.4 Prototypen

Um eine abschließende Bewertung der evaluierten Frameworks vornehmen zu können, wird zunächst mit jedem der vorgestellten Frameworks ein minimaler Web Service implementiert. In dieser Phase soll festgestellt werden, welche der Frameworks besser für die Integration in ein bestehendes System geeignet sind als andere. Die ersten Prototypen, die mit Hilfe der jeweiligen Frameworks umgesetzt werden, sind als eigenständige Projekte und ohne Datenbankanbindung oder weitere Integration implementiert. Es geht in dieser Phase

[1] http://aws.amazon.com/

zunächst darum, einen Einblick in die Mechanismen der einzelnen Frameworks zu gewinnen. Auf Basis dieser Prototypen soll entschieden werden, welches Framework sich am Besten für eine Integration in das CommSy eignet.

Der in der zweiten Phase implementierte Prototyp verwendet nur noch das Framework, für das sich auf Basis des ersten Prototypen entschieden wurde und dient der Verifikation dieser Entscheidung. Der Prototyp wird vollständig in das CommSy eingebunden, um die Integrierbarkeit zu testen. Nach wie vor ist in dieser Phase allerdings keine Verwendung der Datenbankanbindung oder anderer Funktionalität des CommSy geplant. Es wird lediglich geprüft, ob die Integrierbarkeit des Prototypen mit Hilfe des gewählten Frameworks möglich ist.

Um abschließend sicher stellen zu können, dass die gesamte geplante Funktionalität umsetzbar ist, wird im letzten Prototypen der Web Service für eine Kategorie komplett implementiert. Bereits mit Hilfe dieses Prototypen ist dann entscheidbar, inwieweit die Wahl des Frameworks die umsetzbare Funktionalität beeinflusst und ob die Entscheidung für das Framework haltbar ist. Zeigt sich in diesem Prototypen, dass notwendige Funktionalität nicht umsetzbar ist, so wird die Wahl eines alternativen Frameworks erforderlich sein und eine neue Evaluation beginnt wieder mit Phase zwei.

4.5 Zusammenfassung

Neben der Abgrenzung, welche Aspekte einer SOA sinnvoll im Kontext des CommSy umsetzbar sind, ist auch für die umgesetzte Funktionalität ein klarer Rahmen abgesteckt. Mit der ausgewählten Funktionalität wird es möglich sein, auf die wesentlichen Konzepte einer SOA einzugehen. Ein Ausblick auf weitere Funktionalität liefert Anreize für weitere Arbeiten in diesem Zusammenhang. Für die drei geplanten Prototypen ist definiert, welche Fragestellungen sie beantworten können, um die endgültige Implementierung aller Web Services fundiert durchführen zu können.

Kapitel 5

5 Vorgehensweise

Um feststellen zu können ob und in welchem Umfang Refactorings für die Integration der Web Services nötig sind, wird zunächst eine Analyse des CommSy durchgeführt. Als wichtige Voraussetzung für die Implementierung der Web Services gilt es anschließend ein Framework zu wählen. Der Evaluation der verschiedenen Frameworks, die sich in der engeren Wahl befinden, geht eine Erläuterung der Kriterien vorweg, die zur Auswahl herangezogen wurden. Zuletzt folgt die Orchestrierung eines beispielhaften Prozesses, um weitere Merkmale einer SOA diskutieren zu können.

5.1 Architekturanalyse

Um die geplanten Web Services im CommSy implementieren zu können, sind einige Voraussetzungen zu erfüllen. Da im CommSy bereits eine konsequent umgesetzte Schichtenarchitektur zu finden ist, ist nicht mit umfangreichen Refactorings zu rechnen. Die saubere Trennung zwischen Präsentations-, Logik- und Serviceschicht, die es ermöglicht, die Web Services als zusätzlichen Einstiegspunkt neben der Web Oberfläche zu integrieren, ist bereits vorhanden. Im Kontext einer SOA werden an Dienste, die in dieser zur Verfügung gestellt werden, weitere Anforderungen gestellt (vgl. Kapitel 2.4.5, S. 10). Um diesen Anforderungen gerecht werden zu können, muss auch die darunter liegende Schicht entsprechenden Anforderungen genügen. Insbesondere die Forderungen nach Zustandslosigkeit und loser Kopplung der Dienste stehen unter Umständen in Konflikt mit einer Implementierung, die mit Blick auf sessionbasierte Zugriffe durch eine Weboberfläche konzipiert ist. Bei der Analyse des CommSy zeigte sich aber, dass auch hier sehr gute Voraussetzungen für die Implementierung der Web Services gegeben sind.
Die Web Services befinden sich als XML-Repräsentation der Daten des CommSy in der Präsentationsschicht. Anders als die Web Oberfläche greifen die Web Services nicht auf Kommandos aus der Logikschicht zu, sondern verwenden die Implementierungen aus der Serviceschicht direkt. Da sämtliche implementierten Web Services lediglich Funktionen benötigen, die direkt in den Services zur Verfügung stehen, wurde darauf verzichtet, Wrapper-Kommandos in der Logik-Schicht zu implementieren, die diese Aufrufe durchreichen. Da auch für andere Teile des CommSy keine strikte Schichtung gefordert ist, konnte so auf „funktionslose“ Klassen verzichtet werden, die lediglich Aufrufe durch Schichten durchreichen.

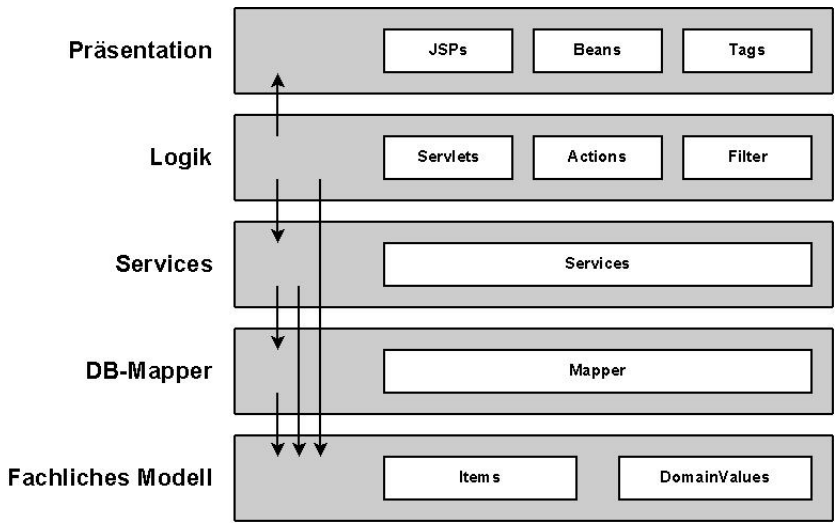

Abbildung 5.1: Schichten vor Einführung der Web Services

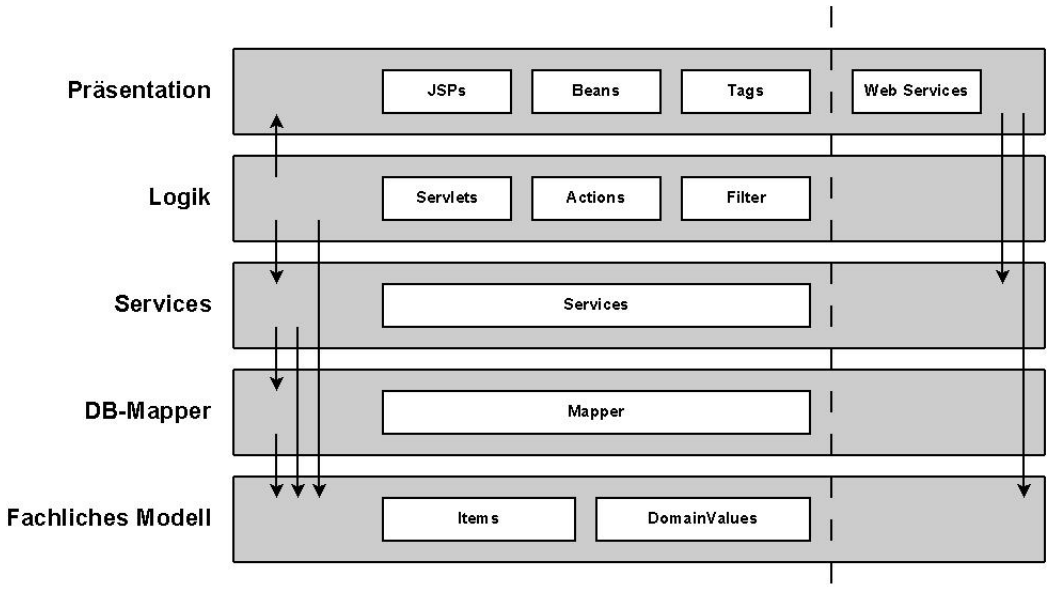

Abbildung 5.2: Schichten nach Einführung der Web Services

5.2 Evaluation des einzusetzenden Frameworks

Um die Entscheidung für eines der zahlreichen Frameworks fällen zu können, die zur Implementierung von Web Services zur Verfügung stehen, sind verschiedene Kriterien zu untersuchen. Dabei gilt es vor allem, den Kontext der Implementierung zu berücksichtigen. Ob ein Web Service eigenständig und von Grund auf neu entwickelt wird, oder in eine bestehende Applikation integriert werden soll, wie es beim CommSy der Fall ist, spielt eine große Rolle. Die Vielzahl der vorhandenen Frameworks ist auf eine solche Integration beispielsweise unterschiedlich gut vorbereitet.

5.2.1 Kriterien für die Auswahl

Bei der Untersuchung der zur Verfügung stehenden Frameworks wurden die folgenden Kriterien zur Bewertung herangezogen. Alle Kriterien wurden insbesondere im Kontext der Implementierung der Web Services für das CommSy beurteilt. Daher spielt eine besondere Rolle, wie groß der Aufwand für die Integration der einzelnen Frameworks ist.

Funktionalität

Im Rahmen der Implementierung für das CommSy ist die geforderte Funktionalität überschaubar. Eine Vielzahl der für Web Services zur Verfügung stehenden Erweiterungen sind für spezielle Szenarien vorgesehen und auch deshalb als Erweiterung konzipiert. Ein Auswahl von Erweiterungen, die auch im Kontext des CommSy interessant sind, wurden im Kapitel „Web Service Technologien und Erweiterungen“ (vgl. Kapitel 2.5, S. 15) vorgestellt. Der Erweiterung WS-Notification wurde keine so große Priorität beigemessen wie beispielsweise WS-Security, da eine entsprechende Implementierung im Rahmen der Diplomarbeit nicht vorgesehen ist.

Performance

Zu Recht wird in den meisten Fällen zunächst ein Maximum an Performance gefordert. Bei genauerer Betrachtung zeigt sich allerdings oft, dass dieses im ersten Moment geforderte Maximum nicht zwingend erforderlich ist. Selbstverständlich muss das gewählte Framework in der Lage sein, Anfragen zeitnah zu beantworten. Realtime Anforderungen sind in der Regel und auch im Kontext des CommSy aber nicht sinnvoll und mit Web Services ohnehin nicht realisierbar. Es werden keine großen Unterschiede bei den einzelnen Frameworks erwartet. Lediglich die REST Implementierung Jersey könnte sich durch den weitaus geringeren Overhead der erzeugten Daten und den Wegfall der De-/Serialisierung der XML-Daten von den anderen Frameworks abheben. Im Rahmen dieser Diplomarbeit sind auch auf Grund der geringen Relevanz keine ausführlichen Untersuchungen der Performance der einzelnen Frameworks geplant. Auch wäre eine entsprechende Untersuchung mit einer Integration aller Frameworks in das CommSy verbunden, die aus zeitlichen Gründen nicht umsetzbar ist.

Skalierbarkeit

Eng verbunden mit der Performance steht auch die Skalierbarkeit nicht im Fokus der Untersuchungen für die Auswahl des Frameworks. Es ist zu erwarten, dass sich die durchschnitt-

liche Anzahl der Benutzer, die gleichzeitig auf einem einzelnen CommSy-Server arbeiten, deutlich erhöht, da Clients, die die Web Services automatisiert im Hintergrund verwenden, sehr viel häufiger Zugriffe generieren werden, als ein über die Web Oberfläche angemeldeter Benutzer. Um eine Abschätzung des Einflusses der Web Services auf die Skalierbarkeit des Gesamtsystems vornehmen zu können, reicht aber die Web Service Implementierung alleine nicht aus. Entscheidenden Einfluss auf die generierte Last hat nicht zuletzt die Implementierung der Clients. Welche Web Services und wie häufig diese aufgerufen werden, wirkt sich erheblich auf die Mehrbelastung des Systems pro Benutzer aus. Darüber hinaus wird die Akzeptanz der zur Verfügung stehenden Clients über deren Verbreitung und damit auch über die Anzahl der Benutzern entscheiden, die zusätzlich zur Web Oberfläche Zugriffe auf die Web Services generieren. Je höher der Anteil der Benutzer ist, die zusätzlich mit den Web Services arbeiten, um so geringer wird die Gesamtzahl der Benutzer sein, die auf einem Server gehostet werden können. Die Implementierung der Web Services selbst wirkt sich demnach nicht auf die Skalierbarkeit des Systems aus, da die Web Services als unabhängiger und zusätzlicher Zugangspunkt zu den Daten des CommSy zur Verfügung stehen. Die Web Oberfläche als bestehender Zugang bleibt unverändert bestehen. Die verstärkte Benutzung der Web Services bei Verfügbarkeit entsprechender Clients wird aber die Last pro Nutzer erhöhen und damit die Skalierbarkeit des Systems negativ beeinflussen.

Aufwand der Integration

Da es im Rahmen dieser Diplomarbeit nicht allein um den Aufbau einer SOA bzw. die Implementierung von Web Services geht, sondern das CommSy als komplexe, bestehende Anwendung um SOA-Konzepte erweitert werden soll, fällt dem Aufwand für die Integration der Web Service-Frameworks eine besondere Bedeutung zu. Zentrale Fragestellung ist hier, wie leicht sich Konzepte des Frameworks in die Architektur des bestehenden Systems einbetten lassen. Sind die Vorgaben, die ein Framework macht starr und erzwingen dadurch größere Anpassungen an dem System, in das es integriert werden soll, oder ist das Framework flexibel genug, dass es sich leicht in das bestehende System einbetten lässt? Im Idealfall stimmen Technologien, die im Framework zum Einsatz kommen, mit den Technologien des Systems, in das es integriert werden soll, überein.

5.2.2 Evaluierte Frameworks

Axis 2

Axis2[1] ist eine komplette Neuentwicklung des bereits seit Oktober 2002 zur Verfügung stehenden Axis1 Frameworks. Die mit Axis2 im August 2004 eingeführte Architektur ist nach Aussage der Entwickler deutlich flexibler als die Axis1 Architektur und ermöglicht so eine einfachere Erweiterung des Frameworks. Der Fokus bei Axis2 liegt in der Unterstützung einer möglichst großen Anzahl an Standards und Web Service-Extensions. Der Aufbau von Axis2 ist stark auf das eigenständige Deployment der Web Services ausgerichtet. Die Standardmethode sieht eine eigenständige WebApplikation vor, die in einem überwachten Verzeichnis vorliegende Web Service-Archive (*.aar Dateien) zur Verfügung stellt. Wird ein servletbasiertes Deployment bevorzugt, ähnelt der workflow sehr dem durch das Standalone-Deployment vorgegebenen Muster. Verfügbare Web Services werden zur

[1] http://ws.apache.org/axis2/

Laufzeit evaluiert und die gesamte Klassenhierarchie nach annotierten Klassen durchsucht. Eine Konfiguration der Web Services ist nicht vorgesehen.

CXF

Auch CXF[2] greift als Weiterentwicklung seines Vorgängers XFire[3] auf langjährige Erfahrung zurück. Im Gegensatz zu Axis2 ist es eines der Hauptanliegen der Entwickler von CXF, das Framework möglichst leicht integrierbar zu machen. Daher hebt CXF einen code-first Ansatz für das Design der Web Services hervor. Im Gegensatz zu Axis2 ist bei CXF Servlet basiertes Deployment als Standard vorgesehen. Web Services können über Annotation oder xml-Datei konfiguriert werden. Ein Parsing der Klassen zur Laufzeit entfällt, da CXF die anzubietenden Web Services bereits zum Zeitpunkt des Kompilierens bekannt gemacht werden.

Jersey

Als Alternative zu Axis2 und CXF, die als Web Service-Frameworks eine entsprechend komplexe Umsetzung erfordern, wurde mit Jersey[4] die Referenz-Implementierung des JSR311[5] (JAX-RS: Java API for RESTful Web Services) evaluiert, die dem REST-Standard (vgl. 2.4.6, S. 13) entsprechend durch Einfachheit und geringen Protokolloverhead hervorsticht. Mit der seit September 2008 verfügbaren Version 1.0 des JSR-311 steht eine stabile API zur Verfügung. Die Referenzimplementierung Jersey liegt zum jetzigen Zeitpunkt ebenfalls in Version 1.0 vor und hat laut eigenen Angaben bereits Produktionsreife.

5.2.3 Auswahl und Begründung

Im Kontext des CommSy ist für die Auswahl des Frameworks unter anderem entscheidend, wie leicht es sich in die bestehende Implementierung integrieren lässt. Darüber hinaus galt es zu prüfen, inwiefern die evaluierten Frameworks in der Lage sind, die geforderte Funktionalität (vgl. Kapitel 4.3 S.30) umzusetzen. Da der zusätzliche Einstiegspunkt in das Commsy im Rahmen dieser Implementierung nur resourcenbezogen geplant ist, schien Jersey als Framework sehr interessant, da es sich sehr einfach integrieren lässt und eine leicht zu verwendende API anbietet. Später wünschenswerte Funktionalität wie beispielsweise Benachrichtigungen durch den Server bei neuen Ereignissen in Räumen des CommSy ließen sich mit diesem Framework allerdings nicht realisieren. Auch wäre eine Authentifizierung für jede mit Jersey angebotene Ressource zu implementieren. Erweiterungen der Web Service API (vgl. Kapitel 2.5.5, S. 23) bieten ein Vielfaches der Funktionalität und wurden daher als Voraussetzung angenommen. Die Erweiterung WS-Security übernimmt beispielsweise die Authentifizierung für alle Web Services. Weitere Funktionalität dieser Erweiterung wie Signatur oder Verschlüsselung wären sehr leicht auf Basis der Standards nachzurüsten. Der von CXF favorisierte code-first Ansatz ist im Kontext des CommSy sehr angenehm. Da keine parallele Entwicklung von Client und Serverseite nötig ist, kann das Design der Schnittstellen komfortabel über die Java-Interfaces erfolgen. CXF ist auf Basis

[2] http://cxf.apache.org/
[3] http://xfire.codehaus.org/
[4] https://jersey.dev.java.net/
[5] https://jsr311.dev.java.net/

von Spring entwickelt. Im Gegensatz zu Axis2 ist es innerhalb der CXF Web Services somit erheblich einfacher, die bestehende Resource-Injection von Spring zu nutzen und die Datenbankservices des CommSy zu verwenden. Axis2 unterstützt eine weitaus größere Anzahl von Bindings und Erweiterungen und ist zusätzlich bewusst modular konzipiert, um die Entwicklung eigener Erweiterungen zu ermöglichen, die fehlende Funktionalität beinhalten. Beides spielt im Rahmen des CommSy aber eine eher untergeordnete Rolle. CXF scheint damit für die Implementierung besonders geeignet zu sein. Eine endgültige Entscheidung für das zu verwendende Framework kann allerdings erst nach Implementierung der geplanten Prototypen getroffen werden.

5.3 Definition der Service-Schnittstellen

Um ein möglichst schlankes und übersichtliches Design der Services zu erreichen, wurde ein Web Service für jede im CommSy vorhandene Kategorie implementiert. Bei der Auswahl der zu implementierenden Services und der Definition der Schnittstellen wurde im ersten Schritt die Möglichkeit geschaffen, lesend auf das CommSy zuzugreifen. Dazu erhielt jeder Web Service zunächst zwei Methoden. Einen Getter, der alle Objekte der entsprechenden Kategorie zurückliefert und einen Getter, um mit Hilfe der Id eines Objektes explizit auf dieses zugreifen zu können. Der Portalservice sowie der Roomservice stellen hier eine Ausnahme dar, da sie nur eine Methode enthalten, die alle Portale bzw. alle Räume zurückliefern. Anschließend wurde die parallel erarbeitete Diplomarbeit berücksichtigt, die die Anbindung eines mobilen Endgerätes an das CommSy zum Inhalt hat. Den besonderen Anforderungen des mobilen Bereichs tragen daher die Services Rechnung, die die Möglichkeit bieten, Änderungen abzufragen, die seit einem bestimmten Zeitpunkt stattgefunden haben. Das bei der Übertragung aller Daten, die benötigt werden um den Datenbestand im Client synchron zu halten, erzeugte Datenvolumen wird erheblich reduziert. Diese Methoden sind auch für den Einsatz in Clients außerhalb des mobilen Bereichs interessant, da sie es sehr einfach ermöglichen, einen News-Ticker zur Verfügung zu stellen, der über aktuelle Ereignisse im CommSy informiert. Um die Weboberfläche des CommSy konsistent zu halten, wurden Methoden eingefügt, mit deren Hilfe Ankündigungen, Termine, Diskussion und Materialien als gelesen markiert werden können. Ein Client der die Web Services verwendet und dem Benutzer die Möglichkeit gibt, diese Objekte zu betrachten, kann somit den Status der Objekte aktualisieren. Dem Benutzer werden diese beim Anmelden über die Web-Oberfläche dann nicht erneut als ungelesen angezeigt. In weiteren Schritten wurden einige Services um Methoden erweitert, mit deren Hilfe weitere Funktionen des CommSy verwendet werden können. Dazu zählt der Schreibzugriff für ausgewählte Materialien, sowie die Möglichkeit, sich für Benachrichtigungen bei Änderungen in bestimmten Kategorien anzumelden. Die Web Services für diese Anmeldungen unterstützen die Änderungen, die für die ereignisgesteuerten Benachrichtigungen im Rahmen der Diplomarbeit von Matthias Hager nötig waren.

5.3.1 Orchestrierung von Diensten

Durch Orchestrierung verschiedener Dienste lassen sich komplexe Abläufe mit ihrer gesamten Logik hinter einfachen Schnittstellen verbergen. Dabei birgt die Orchestrierung externer Dienste weitere Herausforderungen gegenüber dem Einbinden eigener Dienste. Der

im Rahmen dieser Diplomarbeit geplante Prozess wird sowohl aus internen als auch aus externen Diensten orchestriert werden. Gemäß der drei angenommenen Komplexitätsstufen wird dabei zunächst ein Dienst erstellt, der nur auf interne Dienste zurückgreift. Die externen Dienste werden anschließend eingearbeitet, um die sich dadurch ändernden Eigenschaften untersuchen zu können.

Charakteristika von Diensten

Die drei Typen von Diensten unterscheiden sich in ihren Eigenschaften stark voneinander. Während atomare Dienste, die einfache CRUD-Operationen oder Überprüfungen von Rahmenbedingungen für Fachwerte abbilden, nur wenig Prozess-Logik enthalten, bestehen orchestrierte Dienste in der Regel haupstsächlich aus dem Wissen um Prozesse. Die Unterschiede zwischen orchestrierten Diensten, die nur auf interne oder zusätzlich auch auf externe Dienste angewiesen sind, ergeben sich im Wesentlichen durch die verwendeten Datentypen. Werden zusätzlich externe Dienste verwendet, so müssen Datentypen der externen Dienste häufig konvertiert werden, damit sie verwendet werden können. Oftmals werden eigene Konvertierungsdienste in einer SOA zur Verfügung gestellt, da Konvertierungen über einfache Formatänderungen hinausgehen können und unter Umständen Überprüfungen von Rahmenbedingungen oder Anreicherung um weitere Informationen benötigen.

Charakteristika von Clients

Stehen einem Client Dienste zur Verfügung, die lediglich einfache CRUD Operationen abbilden, so ist es nötig, das gesamte Wissen über Prozessabläufe im Client zu implementieren. Dies ist insbesondere dann aufwändig und fehleranfällig, wenn mehrere Clients für beispielsweise unterschiedliche Plattformen implementiert werden sollen oder die gleichen Prozesse als Teilprozess in verschiedenen anderen Prozessen vorkommen. Ist hingegen ein Großteil der Logik und des Prozesswissens bereits in den Web Services enthalten, so können Clients wesentlich schlanker implementiert werden. Dies reduziert nicht nur den Implementierungsaufwand, sondern verringert auch den Wartungsaufwand. Müssen zum Beispiel Prozessabläufe angepasst werden, ohne das sich Schnittstellen oder Datentypen ändern, so kann dies ausschließlich im Web Service passieren. Eine Anpassung der Clients entfällt. Je mehr Logik sich bereits in den Web Services befindet, um so wahrscheinlicher ist es, dass die verwendeten Datentypen komplexer sind und näher an verwendeten Fachwerten orientiert sind. So wird der im Rahmen dieser Diplomarbeit orchestrierte Web Service „Lesung in Österreich“ den Datentyp Buch enthalten, der stärker an dem im Geschäftsprozess verwendeten Datentyp orientiert ist, als an den im CommSy vorhandenen Materialien.

Mögliche Schwierigkeiten

Anhand der Unterscheidung der drei Komplexitätsstufen von Diensten lassen sich auch Schwierigkeiten, die bei der Implementierung dieser Dienste auftreten können, getrennt untersuchen. Atomare Dienste, die unabhängig von anderen Diensten implementiert sind, spielen dabei keine Rolle. Orchestrierte Dienste, die nur interne Dienste aus der eigenen SOA verwenden, erhöhen die Abhängigkeit und Verantwortung dieser Dienste. Schnittstellenänderungen an Diensten können, wenn diese häufig verwendet werden, zu erheblichem

Aufwand bei der Anpassung anderer Dienste in der gesamten SOA führen. Einer Reihe weiterer Herausforderungen hat man sich zu stellen, wenn auch externe Dienste in die Orchestrierung mit einbezogen werden. Zunächst unterliegen die externen Dienste nicht mehr der eigenen Verantwortung. Sowohl deren Verfügbarkeit als auch die Schnittstellen der Dienste können nicht mehr garantiert werden. Die Funktionsfähigkeit der eigenen Dienste wird dadurch zusätzlich gefährdet. Auch schon während der Orchestrierung kann die Einbindung externer Dienste schwieriger umzusetzen sein, als die von internen Diensten. So stellen sich verwendete Datentypen sehr häufig als unterschiedlich heraus. Zusätzliche Konvertierungsmechanismen sind nötig, um die Verwendung der externen Dienste zu ermöglichen. Dabei kann sich unter Umständen herausstellen, dass eine Verwendung unmöglich ist, da externe Fachwerte um Information angereichert werden müssen, die den eigenen Diensten nicht zur Verfügung steht. Neben den eigentlichen Diensten findet man daher in umfangreicheren SOAs neben einem Verzeichnis der angebotenen Dienste auch ein Verzeichnis der verwendeten Datentypen und Methoden, die die Konvertierung von Datentypen ermöglichen.

5.3.2 Einbindung externer Dienste

Mittlerweile stehen eine ganze Reihe von freien und kostenpflichtigen Diensten zur Verfügung, die mit Hilfe der unterschiedlichstens Technologien angeboten werden. Für die Integration in den Prozess „Lesung in Österreich“ wurden beispielhaft zwei Web Services ausgewählt, die kostenlos zur Verfügung stehen und eine einfache Integration ermöglichen.

ISBN prüfen

Von daehosting.com[6] wird ein Web Service zur Verfügung gestellt, der die Überprüfung einer ISBN ermöglicht. Dabei werden keine Informationen über das repräsentierte Buch zur Prüfung der ISBN herangezogen oder durch den Web Service zurückgegeben. Die beiden Methoden IsValidISBN13 und IsValidISBN10 überprüfen lediglich das Format und die Prüfsummer der eingegebenen ISBN.

Adressvalidierung (Österreich)

Um die für eine Veranstaltung eingegebene Adresse zu validieren, wird der von Cortexsoft[7] kostenlos angebotene Web Service[8] verwendet. Dieser ermöglicht die Überprüfung, ob Ort und Postleitzahl der eingegebenen Adresse gültig sind. Leider stand zum Zeitpunkt der Implementierung ein vergleichbarer Service für Deutschland nicht zur Verfügung, so dass die Umsetzung nur für Adressen in Österreich möglich war.

5.4 Zusammenfassung

Bereits die erste Analyse des CommSy und die Vorbereitungen für die zu implementierenden Prototypen, auf deren Basis die Entscheidung für das einzusetzende Web Service Framework basieren wird, zeigen deutlich, in welche Richtung es gehen wird. Wie erwartet

[6] http://webservices.daehosting.com/services/isbnservice.wso?WSDL
[7] http://cortexsoft.com/info/addressvalidierung/
[8] http://87.230.19.126/ValidAddressWebServiceService/ValidAddressWebService?wsdl

werden kaum Refactorings am CommSy nötig sein und auch durch die technologische Nähe zeichnet sich CXF bereits jetzt als Favorit für die Implementierung ab. Die Untersuchung der zu berücksichtigenden Kriterien für die Definition der Schnittstellen und die Auswahl der umzusetzenden Funktionalität wird es später ermöglichen (vgl. Kapitel 6.3.2, S. 47), die Schnittstellen der Web Services zu entwerfen.

Kapitel 6

6 Implementierung der Web Services

Entsprechend den Erkenntnisse, die durch die Prototypen erlangt werden konnten, wurde die Implementierung der Web Services mit Hilfe des CXF Frameworks durchgeführt. Neben den umgesetzten Prototypen findet sich in diesem Kapitel auch die Dokumentation der internen Web Services sowie des orchestrierten Dienstes „Lesung in Österreich“.

6.1 Eingesetzte Software

Für die Implementierung der Prototypen und der Integration der Web Services im CommSy kam folgende Software in den angegebenen Versionen zum Einsatz:

- Eclipse Version 3.4.1
- Apache Tomcat Version 6.0.18
- Apache CXF Version 2.1.3
- Apache Axis2 Version 1.4.1
- Jersey 1.0
- Restlet 1.0.1

6.2 Prototypen

Zunächst wurde mit Hilfe jedes in der engeren Wahl befindlichen Frameworks ein eigenständiger Web Service implementiert (vgl. Kapitel 4.4, S. 32). Auf Basis dieser Prototypen konnte festgestellt werden, dass wie bereits angenommen eine Integration von CXF in das CommSy besonders leicht funktionieren würde, da beide Systeme auf Spring basieren.

In dem darauf folgenden Prototypen wurde nur noch auf das zur Implementierung ausgewählte Framework CXF zurückgegriffen. Es wurde in dieser Phase der bereits umgesetzte Prototyp in das CommSy integriert, um die Annahme zu bestätigen, dass die technologische Verwandschaft von CXF und CommSy dies erheblich vereinfachen würde. Der Web

Service, der nun im CommSy integriert war, benutzte in diesem Stadium nach wie vor keine Funktionalität aus dem CommSy. Als Teil dieses Prototypen wurde bereits die Erweiterung WS-Security verwendet, ohne diese allerdings mit den realen Benutzerkonten aus der Datenbank zu verbinden.

Um abschließend sicher stellen zu können, dass die gesamte geplante Funktionalität umsetzbar ist, wurde im letzten Prototypen der Web Service für die Kategorie „Ankündigungen“ komplett implementiert. In dieser Phase stellte sich besonders eindrucksvoll heraus, wie wichtig die Entscheidung für den Einsatz des richtigen Frameworks war. Da sowohl CXF als auch das CommSy Spring für resource injection einsetzen, war es mit sehr geringem Aufwand möglich aus den umgesetzten Web Services direkt auf die Services des CommSy und somit auf die benötigte Funktionalität zuzugreifen. Die Wahl eines anderen Frameworks hätte hier zu erheblich größerem Aufwand geführt.

6.3 Implementierung der Web Services

Die verschiedenen Kategorien des CommSy wurden in einzelnen Web Services abgebildet, die im Folgenden genauer beschrieben werden. Gemäß der ausgewählten Funktionalität (vgl. Kapitel 4.3, S. 30) finden sich lesende Operationen in allen Web Services. Ausgewählte Kategorien erhielten zusätzliche Methoden. Alle Web Services erfordern eine Authentifizierung über die WS-Security Erweiterung.

6.3.1 WS-Security

Um die Authentifizierung mit Hilfe der WS-Security Erweiterung durchzuführen müssen die entsprechenden Header in der SOAP-Nachricht gesetzt werden. Eine Erläuterung der WS-Security Header ist im Kapitel 2.5.4 (siehe S. 21) zu finden.

Im Rahmen der Diplomarbeit wurden zwei Varianten zur Authentifizierung implementiert. Wird PasswordText als Authentifizierungsmethode gewählt, so wird das Passwort im Klartext an den Server übertragen. PasswordDigest hingegen erzeugt einen Hash, der übertragen wird. Da die Transportsicherheit und die Authentifizierungsmethode bei der WS-Security Methode austauschbar sind, muss der WS-Security callback das Klartextpasswort an das backend übergeben. Die Passwörter des CommSy befinden sich als md5-hash in der Datenbank und nicht in einem KeyStore. Somit kann der Server nicht auf die Klartextpasswörter zugreifen. Daher ist es, wenn PasswortDigest als Passworttyp gewählt wird, auf Clientseite nötig, den md5-hash des Passworts zur Berechnung des Digest heranzuziehen. Für den Server stellt in diesem Falle der md5-hash aus der Datenbank das Passwort dar.

Wird PasswordText als Authentifizierungsmethode gewählt und somit das Passwort im Klartext in der SOAP-Nachricht übertragen, so sollte unbedingt auf ausreichende Transportsicherheit, beispielsweise durch den Einsatz von HTTPS, geachtet werden.

6.3.2 Interfaces

Entsprechend der in Kapitel 5.3 (siehe S. 40) getroffenen Auswahl und Priorisierung der Funktionalität wurden die Schnittstellen der einzelnen Web Services definiert. Dabei wurde wie geplant ein Web Service pro Kategorie des CommSy umgesetzt.

AnnouncementWebService

Mit Hilfe des AnnouncementWebService lassen sich die wesentlichen Funktionen aus der Kategorie „Ankündigungen“ ausführen. Es steht allerdings kein Schreibzugriff über diesen Web Services zur Verfügung. Das Ändern oder Hinzufügen von Ankündigungen ist daher nicht mögliche. Lediglich der Gelesen-Status einer Ankündigung kann gesetzt werden.

getAnnouncement(int) Ist die Id einer Ankündigung bekannt, kann diese gezielt mit der Methode getAnnouncemnet abgefragt werden.

getAnnouncements(int, int) Um die in einem Raum verfügbaren Ankündigungen zu erhalten, kann die Methode getAnnouncements aufgerufen werden. Diese Methode liefert alle für diesen Raum verfügbaren Ankündigungen zurück. Die Methode sollte möglichst nur zur Initialisierung des Clients verwendet werden. Die Anzahl der zu übertragenden Daten und damit auch die Serverlast kann durch Verwenden der Methode getModifiedAnnouncementsSince reduziert werden.

getModifiedAnnouncementsSince(int, int, Date) Um die Anzahl der Ankündigungen zu minimieren und eine Mehrfachübertragung der Ankündigungen zu vermeiden, kann der Service getModifiedAnnouncementsSince aufgerufen werden. Dabei wird das Datum der letzten Abfrage übergeben. Der Server liefert dann nur die Ankündigungen zurück, die sich seit diesem Zeitpunkt geänderten haben oder neu hinzugekommenen sind.

markAnnouncementRead(int) Die Methode markAnnouncementRead ermöglicht es, einzelne Ankündigungen als gelesen zu markieren. Ein die Web Services verwendender Client kann und sollte damit sicherstellen, dass der Status von Ankündigungen aktualisiert wird und dem Benutzer beim Verwenden der Web-Oberfläche keine Nachrichten, die er bereits gelesen hat, als neu bzw. ungelesen präsentiert werden.

AppointmentWebService

Analog zu den bereits im AnnouncementWebService beschriebenen Methoden enthält auch der AppointmentWebService vier Methoden, die die Standardoperationen abbilden:

- getAppointment(int)
- getAppointments(int, int)
- getModifiedAppointmentsSince(int, int, Date)
- markAppointmentRead(int)

Darüber hinaus ist es möglich, mit Hilfe der Methode addAppointment(int, int, AppointmentDAO) einen Termin hinzuzufügen.

DiscussionWebService

Der DiscussionWebService enthält ebenfalls vier Methoden, die in allen Kategorien vorhanden sind:

- getDiscussion(int)
- getDiscussions(int, int)
- getModifiedDiscussionsSince(int, int, Date)
- markDiscussionRead(int)

In dieser Kategorie sind darüber hinaus noch weitere Methoden implementiert.

addDiscussion(int, int, DiscussionDAO) fügt eine Diskussion einem ausgewählten Raum hinzu.

getDiscussionArticles(int, int, int) liefert alle zu einer Diskussion gehörenden Artikel zurück.

getModifiedDiscussionArticlesSince(int, int, int, Date) iefert alternativ nur die Artikel einer Diskussion zurück, die sich seit dem angegebenen Datum geändert haben.

MaterialWebService

Im MaterialWebService sind ebenfalls nur die vier Methoden enthalten, die die Standardoperationen abbilden:

- getMaterial(int)
- getMaterials(int, int)
- getModifiedMaterialsSince(int, int, Date)
- markMaterialRead(int)

Auch hier ist es möglich, mit Hilfe der Methode addMaterial(int, int, MaterialDAO) ein Material in dem ausgewählten Raum hinzuzufügen.

PortalWebService

Über diesen Web Service kann die Liste der zur Verfügung stehenden Portale abgefragt werden. Er enthält lediglich die Methode getPortals().

RoomWebService

Über diesen Web Service können die verfügbaren Räume eines Portals abgefragt werden. Ein Hinzufügen oder Bearbeiten von bestehenden Räumen ist nicht möglich.

getRooms() liefert eine Liste aller Räume im Portal, auf der authentifizierte Benutzer Zugriffsberechtigungen hat.

subscribeRoom(int, int, boolean) bietet die Möglichkeit, den Benutzer für Benachrichtigungen eines Raumes an- bzw. abzumelden.

getSubscribedRooms(int) liefert die Liste aller abonnierten Räume eines Benutzers für ein bestimmtes Portal zurück.

UserWebService

Der UserWebService liefert über die Methode getUser(int) Benutzerinformationen zu dem Benutzer mit der übergebenen ID zurück. Dies wird insbesondere dann benötigt, wenn zusätzliche Informationen zu einem Benutzer benötigt werden, der über die ID in anderen Objekten referenziert ist.

6.3.3 Data Access Objects

Um die Web Services frei von Typen-Deklaration zu halten, die nicht benötigt werden und damit auch die entstehenden Schnittstellen sowie den Datenverkehr zu minimieren, wurden für alle Typen, die in den Web Services verwendet werden, Data Access Objects erstellt.

6.3.4 Spring Resource Injection

Da sowohl im CommSy selbst als auch im für die Web Services verwendeten Framework Spring zum Einsatz kommt, ist es sehr komfortabel möglich, die von Spring zur Verfügung gestellte resource injection für den Zugriff auf die Datenbankservices des CommSy innerhalb der Web Services zu verwenden. Sowohl in dem für die WS-Security Erweiterung nötigen CallBack, der die Authentifizierung abwickelt, als auch in den Web Services für die einzelnen Kategorien des CommSy ist somit nur Quellcode enthalten, der die jeweilige Funktionalität betrifft. Es ist an keiner Stelle zusätzlicher Code nötig, um an die Resourcen des CommSy zu gelangen.

6.3.5 Filter

Im CommSy werden alle schreibenden Datenbankzugriffe innerhalb von Transaktionen ausgeführt. Um das Öffnen und Schließen der Transaktionen nicht innerhalb jeder Service-Implementierung durchführen zu müssen, sind entsprechende Filter implementiert. Der CommsyWebserviceFilter implementiert das Transaktionsverhalten für alle Web Services.

6.3.6 Abstract Web Service

Die abstrakte Implementierung AbstractWebServiceImpl enthält zwei Methoden, die in jedem Web Service benötigt werden. Die Methode getUsername extrahiert den Benutzernamen aus den WS-Security Kopfzeilen. Da die WS-Security Erweiterung vorsieht, dass eine CallBack-Implementierung für die Authentifizierung zuständig ist, kann in den Web

Services davon ausgegangen werden, dass der Benutzer, der von dieser Methode zurückgegeben wird, bereits authentifiziert ist. Schlägt die Authentifizierung im CallBack fehl, so werden die Web Service Methoden nicht aufgerufen.

Als zweite Helfermethode steht fillItemDAO zur Verfügung. Sie dient lediglich dem einfachen Auffüllen der an allen von Item erbenden Klassen vorhandenen Member.

6.4 Orchestrierung von Diensten

Die bisher umgesetzte Funktionalität der Web Services beschränkt sich auf das Lesen und Hinzufügen von Objekten des CommSy. Das Ändern von Objekten war im Rahmen dieser und der parallel stattfindenden Diplomarbeiten nicht nötig und wurde daher aus Zeitgründen nicht umgesetzt. Um jedoch auf die weiteren Merkmale einer SOA eingehen zu können, wurde der Prozess „Lesung in Österreich“ beispielhaft für das Zusammenspiel von mehreren Diensten in einem orchestrierten Dienst umgesetzt. Bei der Orchestrierung wurden sowohl im Rahmen des CommSy implementierte Web Services als auch extern verfügbare Web Services verwendet, um verschiedene Schwierigkeiten, die dabei unter Umständen auftreten, diskutieren zu können.

6.4.1 Prozess: Lesung in Österreich

Beim Anlegen einer Veranstaltung „Lesung in Österreich“ im CommSy gilt es einige Besonderheiten zu beachten. Neben der Tatsache, dass diese Treffen in Österreich stattfinden, ist es nötig, dass ein Buch während der Veranstaltung zur Verfügung steht. Wird eine Lesung in Östereich angelegt, so muss daher zunächst geprüft werden, ob der Veranstaltungsort in Österreich liegt und eine gültige Postleitzahl hat. Ist dies der Fall, muss zusätzlich geprüft werden, ob das Buch, dem die Veranstaltung gewidmet ist, überhaupt existiert. Daher wird die angegebene ISBN-Nummer geprüft. Erst wenn diese Voraussetzungen erfüllt sind, kann die Veranstaltung angelegt werden. Die beiden für die Orchestrierung verwendeten, externen Dienste stellen rudimentäre Funktionalität kostenlos zur Verfügung. Sowohl die Überprüfung einer Adresse, als auch die Plausibilitätsprüfung der Buchdaten könnte durch entsprechend aufwändigere Dienste umfassender implementiert werden.

6.4.2 Verwendete interne Dienste

Um den Prozess „Lesung in Österreich“ umsetzten zu können, werden zwei interne Dienste benötigt. Zum einen wird die Veranstaltung als „Appointment“ mit Hilfe der Methode addAppointment(int, int, AppointmentDAO) des AppointmentWebService angelegt. Da die Methode zum Erzeugen einer Lesung im selben Web Service angesiedelt wurde, ist dies durch einen einfachen Methodenaufruf möglich. Hier zeigte sich bereits, wenn auch nur sehr rudimentär, das eine Konvertierung der Datentypen notwendig ist, wenn verschiedene Anforderungen gegeben sind. Ein Appointment des CommSy enthält als Ortsangabe lediglich einen String. Der Prozess zum Anlegen einer Lesung sieht jedoch vor, dass Postleitzahl und Ort validiert werden müssen. Der Datentyp für die Lesung behandelt Postleitzahl und Ort demnach getrennt. Da beide Datentypen selbst verwaltet sind, wurde die Konvertierungsmethode im Datentyp „LectureDOA“ selbst untergebracht. Dieser bietet eine Methode

getAsAppointmentDAO(), um den für das CommSy benötigte Datentyp zu erzeugen. Bei dieser Konvertierung werden lediglich Postleitzahl und Ort zusammen im einzelnen String des AppointmentDAO für den Ort gespeichert. Obwohl diese Konvertierung so einfach ist, zeigt sie ein weiteres Problem, das auftreten kann. Eine Konvertierung von einer so im CommSy angelegten Lesung, die als Appointment gespeichert wurde, zurück zum Datentyp, der im Prozess „Lesung in Österreich“ zum Einsatz kommt, ist nicht mit ausreichender Sicherheit zu gewährleisten. Das Anbieten weiterer Dienste im Zusammenhang mit den so gespeicherten Lesungen ist damit nur eingeschränkt möglich.

Der zweite interne Dienst, der für die Umsetzung der Orchestrierung verwendet wird, ist der MaterialWebService. Das zu der Veranstaltung gehörende Buch wird mit Hilfe der Methode addMaterial(int, int, MaterialDAO) als Material angelegt. Neben der auch hier nötigen Konvertierung des Datentyps BookDAO zu einem Material des CommSy zeigt sich an dieser Stelle eine weitere Schwierigkeit, die bei der Orchestrierung auftreten kann. In späteren Lebenszyklen einer SOA werden neu umzusetzende Prozesse unter Umständen Funktionalität erfordern, die durch die vorhandenen Dienste nicht abbildbar ist. Wird also in der ersten Phase der Implementierung einer SOA nicht die gesamte Funktionalität des Basissystems durch Dienste zur Verfügung gestellt, so ist es sehr wahrscheinlich, dass diese im Laufe der Zeit nachträglich umgesetzt werden muss. Im Kontext dieser Arbeit zeigt sich das durch die Notwendigkeit, das angelegte Buch mit dem Termin zu verknüpfen. Bei der Planung der Web Services, die für die Implementierung vorgesehen sind, wurde eine solche Verknüpfung nicht berücksichtigt. Die Auswahl der umgesetzten Funktionalität richtete sich mit unter an der parallel stattfindenden Arbeit von Matthias Hager aus (vgl. Kapitel 4.3, S. 30) die nur den Lesezugriff auf die Daten des CommSy benötigt. Der später konzipierte Prozess „Lesung in Österreich“ stellt also zusätzliche Anforderungen, die durch die umgesetzten Web Services nicht abgebildet werden können. Leider zeigte sich dieses Defizit erst sehr spät, so dass ein konsequente Lösung dieses Problems durch Implementierung eines weiteren Web Services, der das Verknüpfen von Daten im CommSy ermöglicht, aus zeitlichen Gründen nicht mehr umsetzbar war. Da ein Verknüpfen vom Objekten im CommSy aber bereits von einem eigenen Service durchgeführt wird, wäre die Implementierung eines solchen Web Services technisch umsetzbar. Als Übergangslösung wurde durch das Ergänzen des Buchtitels zur Beschreibung des Termins im CommSy die Zusammengehörigkeit zumindest für den Benutzer erkennbar gemacht.

6.4.3 Eingebundene externe Dienste

Die beiden externen Web Services, die im orchestrierten Web Service für den Prozess „Lesung in Österreich“ zum Einsatz kommen, wurden mit Hilfe des Axis Frameworks eingebunden. Dieses wurde für die Implementierung der Web Service clients auch deshalb gewählt, weil die Unabhänigkeit von zur Implementierung verwendeter Technologie geprüft werden sollte. Da auch Axis im CommSy bereits an anderer Stelle zum Einsatz kommt, entstanden keine zusätzlichen Abhängigkeiten. Bei der Implementierung der externen Dienste traten keine SOA-spezifischen Schwierigkeiten auf. Die im orchestrierten Dienst verwendeten Datentypen sind darauf abgestimmt, alle für die Verwendung der externen Dienste benötigten Informationen zu enthalten. Eine Konvertierung ist an dieser Stelle also nicht nötig. Nennenswert ist im Zusammenhang mit den eingebundenen Diensten allerdings die

so erzeugte Abhängigkeit. Änderungen an diesen Diensten werden den eigenen Dienst unbenutzbar machen bis entsprechende Anpassungen umgesetzt sind. Dies mag im vorliegenden Beispiel noch unwahrscheinlich und ggf. mit geringem Aufwand umsetzbar sein, kann aber in aufwändigeren Szenarien durchaus problematische Ausmaße erreichen. Auch schon in diesem Beispiel ist natürlich die Verfügbarkeit der externen Dienste ein kritischer Faktor, der nicht selbst gewährleistet werden kann.

6.5 JUnit-Tests

Für die implementierten Web Services wurden jeweils eigene JUnit-Tests umgesetzt, um deren Funktionalität zu gewährleisten. Insbesondere für den orchestrierten Dienst werden hier auch Fehlerfälle überprüft. Um die eingebundenen Dienste zu prüfen, wird eine fehlerhafte ISBN und eine ungültige Kombination aus Postleitzahl und Ort getestet. Für alle weiteren Methoden sind positive Funktionstests implementiert.

6.6 Zusammenfassung

Die Integration der Web Services im CommSy war auf Grund der Wahl von CXF als Web Service Framework ohne größere technische Schwierigkeiten möglich. Die sauber umgesetzte Schichtenarchitektur im CommSy und die konsequente Verwendung von Spring ermöglichen eine nahtlose Zusammenarbeit der Services im CommSy mit den neuen Web Services. Auch die Implementierung des orchestrierten Dienstes brachte die erwarteten Ergebnisse. Verschiedene Herausforderungen und Schwierigkeiten aus dem SOA-Umfeld konnten an dieser Stelle beispielhaft demonstriert werden. Die Implementierung setzt die geplante Funktionalität stabil um und bildet eine solide Basis für weitere Entwicklungen in diesem Kontext.

Kapitel 7

7 Analyse der erzielten Ergebnisse

Die Untersuchungen sowohl am CommSy selbst, als auch der Charakteristika von Diensten in einer SOA allgemein, sowie den Herausforderungen, denen man sich bei der Integration eines bestehenden Softwaresystems in eine SOA zu stellen hat, haben gezeigt, dass eine SOA nicht nur Vorteile mit sich bringt. Die seit einiger Zeit in den Medien und vor allem in der Werbung kursierenden Begriffe für eine Vielzahl von neuen Produkten, die scheinbar alle Probleme der IT zu lösen vermögen, aus dem SOA-Umfeld sind nicht unkritisch zu betrachten. Den angepriesenen Vorteilen stehen ein unter Umständen hoher Integrationsaufwand und die umfangreiche Verwaltung der Dienste und Datentypen gegenüber. Es ist also neben der Frage, mit welchen Technologien eine SOA aufgebaut werden soll, zunächst einmal die Frage zu stellen, ob es im vorliegenden Kontext überhaupt sinnvoll ist, eine SOA aufzubauen. Die im Rahmen dieser Diplomarbeit durchgeführte Integration der Web Services im CommSy kann nur ansatzweise an die Dynamik herankommen, die in einer komplexen SOA entsteht. Einige der möglichen Schwierigkeiten sind durch den beispielhaft implementierten Prozess „Lesung in Österreich“ angedeutet. Trotz des zu erwartetenden Aufwandes sowohl bei der Integration als auch bei der Pflege kann die Implementierung einer SOA zu einer erheblichen Produktivitätssteigerung führen. Diese wird allerdings nicht zwangsläufig erreicht und es gilt im Einzelfall genau zu prüfen, ob der Integrationsaufwand lohnt.

7.1 Konnten Mehrwerte erzielt werden?

In Kapitel 3 (siehe S. 25) wurden zu Beginn der Arbeit eine Reihe von Mehrwerten vorgestellt, die sich durch Implementierung einer SOA erreichen lassen. Nachfolgende Überlegungen zeigen, in welchem Umfang diese Mehrwerte durch die konkrete Umsetzung im CommSy erreichte werden konnten.

7.1.1 Architektonische Mehrwerte

Die Betrachtung der architektonischen Mehrwerte muss zweigeteilt stattfinden. Zum einen stand zu Beginn der Diplomarbeit die Frage im Raum, ob die Architektur des CommSy selbst den Anforderungen für die geplante Integration genügen würde. Zum Anderen galt die Aufmerksamkeit der Untersuchung der Architektur, die innerhalb einer SOA aufgebaut

wird. Im Laufe der Arbeit hat sich gezeigt, dass die im CommSy vorhandene Schichtenarchitektur ausreicht, um die geplanten Dienste einführen zu können. Auch die erweiterten Anforderungen, die an Dienste in einer SOA gestellt werden, konnte ohne Änderungen an den durch die Service-Schicht im CommSy zur Verfügung gestellten internen Services, erfüllt werden. So bewahrheitete sich die Vermutung nicht, dass der Fokus auf ein session-basiertes Nutzungsmodell der Web-Oberfläche zu einer nötigen Anpassung der Services führen würde. Eine Änderung oder Verbesserung der Architektur des CommSy selbst wurde durch die Einführung der Web Services demnach nicht erreicht. Im Gegensatz zu den internen Qualitätsmerkmalen der Architektur des CommSy konnten aus SOA-Perspektive allerdings durchaus architektonische Mehrwerte erzielt werden. Diese sind im Folgenden genauer beschrieben.

Vorteile durch Plattformunabhängikeit

Da die Daten des CommSy nun plattformunabhängig und in einer maschinenlesbaren Form zur Verfügung stehen, lassen sie sich wesentliche leichter weiter verarbeiten. Ob dies nun bedeutet, dass alternative Clients für den Endnutzer zur Verfügung gestellt werden und damit eine stärkere Unterstützung der unterschiedlichen Vorlieben verschiedener Benutzergruppen geschaffen werden kann, oder eine Integration in andere E-Learning oder Community-Systeme ermöglicht wird, wird sich in Zukunft zeigen müssen. Klar ist, dass durch die Einführung der Web Services die Grundlage geschaffen wurde, um diverse Aspekte umsetzen zu können, die eine SOA interessant machen.

Vorteile durch Zustandslosigkeit

Die Forderung nach Zustandslosigkeit aller Dienste konnte im Rahmen der Web Service Implementierung erfüllt werden. Dieses führt unter anderem zu einer besseren Skalierbarkeit des Serversystems. Die Relationen zwischen den einzelnen Anfragen eines Clients werden aufgehoben. Der Server selbst muss also neben dem Abarbeiten des eigentlichen Requests keine weiteren Ressourcen für das Verwalten der Session zur Verfügung stellen. Bei einer hohen Anzahl von Sessions, die jeweils nur wenige Anfragen ausführen, kann dies zu einer erheblichen Einsparung an Ressourcen führen.

Ist ein Benutzer des Systems regelmäßig daran interessiert, ob neue Informationen zur Verfügung stehen, so muss er, je nach Häufigkeit der Aufrufe, für jede Überprüfung erneut einen Login durchführen und somit eine Session erzeugen. Da die geforderten Informationen durch die Web Services direkt zur Verfügung stehen und die Authentifizierung im Aufruf des Dienstes integriert ist, entfällt der Login-Vorgang und der Aufbau einer neuen Session. Auch dies reduziert die Last des Servers pro Benutzer, wenn Web Services anstatt der Web Oberfläche genutzt werden.

7.1.2 Mehrwerte für den Benutzer

Neben den Verbesserungen, die für das CommSy selbst erreicht werden konnten, ergeben sich auch Mehrwerte für den Benutzer des Systems. Diese sind für den Benutzer allerdings nicht direkt aus der Implementierung der Dienste selbst zugänglich. Erst durch die Implementierung weiterer Softwaresysteme, wie beispielsweise eines alternativen Clients,

der regelmäßig auf Aktualisierungen überprüft, lassen sich Mehrwerte für den Benutzer erzielen.

Vorteile durch Multi-Client-Access

Mit Implementierung der Web Services wurde die Grundlage für eine einfache Anbindung alternativer Clients an das CommSy geschaffen. Durch entsprechende Implementierungen lassen sich eine Vielzahl von Benutzungsmodellen abbilden, die eine Weboberfläche nicht bieten kann. Dies kann zum Einen die Integration der Daten des CommSy in Umgebungen bedeuten, mit denen der Benutzer gewohnt ist zu arbeiten, es lassen sich andererseits aber auch Arbeitsabläufe automatisieren, die dem Benutzer bisher regelmäßige Aktivität abverlangten.

Vorteile durch Dienst-Orchestrierung

Durch die Verfügbarkeit der Funktionalität des CommSy in einzelnen Diensten und die Möglichkeit, diese zu orchestrieren, lassen sich - wie am Beispiel „Lesung in Österreich“ verdeutlicht - komplexe Prozesse kapseln. Dadurch ist es möglich, einzelne, technisch orientiertere Arbeitsschritte, zu Diensten zusammenzufassen, die mehr an den für den Benutzer geläufigen Prozessen orientiert sind. So erleichtert dies, die Einführung entsprechender Dienste vorausgesetzt, die Benutzung des Systems, da dem Benutzer ein Umdenken zwischen der „Prozess-Welt“ und der technischen Umsetzung erspart bleibt. Bei der Einführung der Web Services in das CommSy konnte dies in Ermangelung aufwändiger Prozesse allerdings nicht in ausreichendem Maße umgesetzt werden.

7.2 Welche Probleme traten bei der Bearbeitung auf?

Die Integration der Web Services in das CommSy stellte sich äußerst problemlos dar. Durch die bereits vorhandene Schichtenarchitektur war es sehr leicht möglich, auf für die Web Services nötige Funktionalität im CommSy zuzugreifen. Mit der Entscheidung für CXF als Web Service Framework war das Zusammenspiel der Web Services und der bestehenden Services im CommSy auch auf Grund des konsequenten Einsatzes von Spring sowohl bei CXF als auch im CommSy ohne großen Aufwand möglich. Um mögliche Probleme, die beim Aufbau einer SOA entstehen können, diskutieren zu können, mussten diese durch die Orchestrierung des Prozesses „Lesung in Österreich“ bewusst herbeigeführt werden. Auch wenn ein solcher Prozess nicht Teil der eigentlichen Funktionalität des CommSy ist, konnte so aufgezeigt werden, welche Probleme bei einer über die Implementierung der Web Services hinausgehenden Integration des CommSy in eine SOA auftreten können.

7.3 Wie kann es weiter gehen?

Die Integration der Web Services in das CommSy und die damit verbundenen Untersuchungen in Bezug auf die SOA-fähigkeit eines bestehenden Softwaresystems bedeuten lediglich den Anfang des Aufbaus einer SOA. Aus der hier geschaffenen Grundlage ergeben sich eine Vielzahl von Möglichkeiten, um weitere Aspekte einer SOA umzusetzen. Neben der bereits in Kapitel 4.3 (siehe S. 30) aufgeführten Funktionalität, die aus zeitlichen Gründen

nicht im Rahmen dieser Diplomarbeit umgesetzt werden konnte, bieten auch Projekte außerhalb des CommSy selbst viel Raum für Entwicklung. Zentrale Aspekte einer SOA, wie beispielsweise die Auffindbarkeit der Dienste für andere Konsumenten, sind bisher außer Acht gelassen worden. Dass die Dienste für die Verwendung innerhalb eines Clients für Endnutzer ausreichen, zeigt auch die parallel erarbeitete Diplomarbeit von Matthias Hager. Ob auf Basis der vorhandenen Web Services ein Zusammenspiel mit anderen Community oder E-Learningsystemen möglich ist, um beispielsweise einen automatisierten Datenabgleich umzusetzen, könnte Inhalt weiterer Arbeiten sein.

7.4 Fazit

Im Rahmen dieser Diplomarbeit wurde sowohl die geplante Umsetzung der Web Services, als auch die damit verbundenen Untersuchungen der Eigenschaften einer SOA erfolgreich abgeschlossen. Die zum CommSy beigesteuerte Funktionalität stellt nicht nur eine solide Basis für die Anbindung alternativer Clients dar, sondern geht mit den angebotenen Diensten auch erste Schritte in Richtung Aufbau einer SOA. Ob die sich so ergebenden Möglichkeiten genutzt werden, oder der Fokus auf der Kernfunktionalität des CommSy bleibt, muss sich zeigen.

Literaturverzeichnis

[iee 1990] (1990). *IEEE standard glossary of software engineering terminology*.

[iee 2000] (2000). *IEEE Recommended practice for architectural description of software-intensive systems*. http://ieeexplore.ieee.org/servlet/opac?punumber=7040.

[BARISIC et al. 2007] BARISIC, DANIEL, M. KROGMANN, G. STROMBERG und P. SCHRAMM (2007). *Making Embedded Software Development More Efficient with SOA*. In: *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, Bd. 1, S. 941–946.

[BASS et al. 2003] BASS, LEN, P. CLEMENTS und R. KAZMAN (2003). *Software Architecture in Practice*. Addison-Wesley Longman, Amsterdam, 2. A. 2003. Aufl.

[BLANVALET et al. 2006] BLANVALET, STANY, J. BOLIE und M. CARDELLA (2006). *Bpel Cookbook: Best Practices for Soa-Based Integration and Composite Applications Development: Best Practices for SOA-based Integration and Composite Applications Development*. Packt Pub.

[BURBIEL 2007] BURBIEL, HERBERT (2007). *SOA & Webservices in der Praxis*. Franzis, 1 Aufl.

[CERAMI 2002] CERAMI, ETHAN (2002). *Web Services Essentials: Distributed Applications with XML-RPC, Soap, UDDI and Wsdl*. O'Reilly Media, Student. Aufl.

[CLEMENTS et al. 2003] CLEMENTS, PAUL, F. BACHMANN, L. BASS, D. GARLAN, J. IVERS, R. LITTLE, J. STAFFORD, R. NORD und P. CLEMENTS (2003). *Documenting Software Architectures*. Addison-Wesley Longman, Amsterdam.

[DIERCKS und BORN 2007] DIERCKS, J. und A. BORN (2007). *Last Exit SOA*. IX - Magazin für professionelle Informationstechnik, S. 46–51.

[ERL 2007] ERL, THOMAS (2007). *SOA Principles of Service Design*. Prentice Hall International, 1 Aufl.

[ERL 2008] ERL, THOMAS (2008). *SOA*. Addison-Wesley, München, 1 Aufl.

[FIELDING 2000] FIELDING, ROY THOMAS (2000). *Architectural styles and the design of network-based software architectures*. Doktorarbeit, University of California, Irvine.

[FOWLER 2005] FOWLER, MARTIN (2005). *Refactoring- Studentenausgabe. Oder wie Sie das Design vorhandener Software verbessern*. Addison-Wesley, München.

[FRÖSCHLE und REINHEIMER 2007] FRÖSCHLE, HANS-PETER und S. REINHEIMER (2007). *Serviceorientierte Architekturen*. Dpunkt Verlag, 1 Aufl.

[GAMMA et al. 2004] GAMMA, ERICH, R. HELM und R. E. JOHNSON (2004). *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, München.

[GUDGIN et al. 2007a] GUDGIN, MARTIN, M. HADLEY, N. MENDELSOHN, J.-J. MOREAU, H. F. NIELSEN, A. KARMARKAR und Y. LAFON (2007a). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*.

[GUDGIN et al. 2007b] GUDGIN, MARTIN, M. HADLEY, N. MENDELSOHN, J.-J. MOREAU, H. F. NIELSEN, A. KARMARKAR und Y. LAFON (2007b). *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*.

[HAROLD et al. 2001] HAROLD, ELLIOTTE R., W. S. MEANS und W. S. MEANS (2001). *XML in a Nutshell*. O'Reilly.

[HAU et al. 2008] HAU, THORSTEN, N. EBERT, A. HOCHSTEIN und W. BRENNER (2008). *Where to Start with SOA: Criteria for Selecting SOA Projects*. In: *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, S. 314.

[HUVAR et al. 2008] HUVAR, MARTIN, T. FALTER, T. FIEDLER und A. ZUBEV (2008). *Anwendungsentwicklung mit Enterprise SOA*. Galileo Press, 1 Aufl.

[KERIEVSKY 2006] KERIEVSKY, JOSHUA (2006). *Refactoring to Patterns*. Addison-Wesley, München.

[KRAFZIG et al. 2004] KRAFZIG, DIRK, K. BANKE und D. SLAMA (2004). *Enterprise SOA. Service Oriented Architecture Best Practices*. Prentice Hall International, 2005 Aufl.

[KRAFZIG et al. 2007] KRAFZIG, DIRK, K. BANKE und D. SLAMA (2007). *Enterprise SOA. Best Practices für Serviceorientierte Architekturen - Einführung, Umsetzung, Praxis*. Mitp-Verlag, 1 Aufl.

[KUHRMANN und BENEKEN 2006] KUHRMANN, MARCO und G. BENEKEN (2006). *WindowsÂ® Communication Foundation: Konzepte - Programmierung - Migration*. Spektrum Akademischer Verlag.

[LAURENT et al. 2001] LAURENT, SIMON ST, J. JOHNSTON und E. DUMBILL (2001). *Programming Web Applications with XML-RPC*. O'Reilly Media.

[LIEBHART 2007] LIEBHART, DANIEL (2007). *SOA goes real*. Hanser Fachbuchverlag, 1 Aufl.

[LIEGL 2007] LIEGL, PHILIPP (2007). *The Strategic Impact of Service Oriented Architectures*. In: *Engineering of Computer-Based Systems, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops on the*, S. 475–484.

[LOPEZ et al. 2007] LOPEZ, NICOLAS, R. CASALLAS und J. VILLALOBOS (2007). *Challenges in Creating Environments for SOA Learning*. In: *Systems Development in SOA Environments, 2007. SDSOA '07: ICSE Workshops 2007. International Workshop on*, S. 9.

[MACKENZIE et al. 2006] MACKENZIE, C. MATTHEW, K. LASKEY, F. MCCABE, P. F. BROWN und R. METZ (2006). *Reference Model for Service Oriented Architecture 1.0*.

[MAURIZIO et al. 2008] MAURIZIO, AMELIA, J. SAGER, P. JONES, G. CORBITT und L. GIROLAMI (2008). *Service Oriented Architecture: Challenges for Business and Academia*. In: *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, S. 315.

[MELZER 2007] MELZER, INGO (2007). *Service-orientierte Architekturen mit Web Services. Konzepte - Standards - Praxis*. Spektrum Akademischer Verlag, 2. A. Aufl.

[NEWCOMER und LOMOW 2005] NEWCOMER, ERIC und G. LOMOW (2005). *Understanding Service-Oriented Architecture (SOA) with Web Services.*. Addison-Wesley Longman, Amsterdam.

[ORCHARD 2007] ORCHARD, DAVID (2007). *SOAP 1.2 Part 3: One-Way MEP*.

[PILONE und PITMAN 2005] PILONE, DAN und N. PITMAN (2005). *UML 2.0 in a Nutshell: A Desktop Quick Reference (In a Nutshell*. O'Reilly Media, 1 Aufl.

[ROOCK und LIPPERT 2006] ROOCK, STEFAN und M. LIPPERT (2006). *Refactoring in Large Software Projects: Performing Complex Restructurings Successfully*. Wiley & Sons, 1 Aufl.

[SEEBOERGER-WEICHSELBAUM 2000] SEEBOERGER-WEICHSELBAUM, MICHAEL (2000). *Das Einsteigerseminar XML.* BHV Software GmbH & Co. K, 2. Aufl. Aufl.

[SMITH und LEWIS 2007] SMITH, DENNIS und G. LEWIS (2007). *Developing Realistic Approaches for Migration of Legacy Assets to SOA Environments*. In: *Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007. ICCBSS '07. Sixth International IEEE Conference on*, S. 10.

[STARKE und TILKOV 2007] STARKE, GERNOT und S. TILKOV (2007). *SOA-Expertenwissen*. Dpunkt Verlag, 1 Aufl.

[ULLMANN 2004] ULLMANN, ANDREAS (2004). *Was sich hinter SOA verbirgt - Lego für Fortgeschrittene*. Javamagazin, 10:55–61.

[VOGEL et al. 2005] VOGEL, OLIVER, I. ARNOLD, A. CHUGHTAI und M. VÖLTER (2005). *Software-Architektur: Grundlagen - Konzepte - Praxis*. Spektrum Akademischer Verlag, 1 Aufl.

[YEFIRM 2003] YEFIRM, V. NATIS (2003). *Service-Oriented Architecture Scenario*.

[ZHANG 2006] ZHANG, LIANG-JIE (2006). *SOA and Web Services*. In: *Services Computing, 2006. SCC '06. IEEE International Conference on*, S. xxxvi.

Abbildungsverzeichnis

Abkürzungsverzeichnis

Axis2	Apache Axis2/Java - Next Generation Web Services
BPEL	Business Process Execution Language
CORBA	Common Object Request Broker Architecture
CXF	Apache CXF: An Open Source Service Framework
DCOM	Distributed Component Object Model
DTD	Document Type Definition
J2EE	Java 2 Platform Enterprise Edition
JNDI	Java Naming and Directory Interface
OASIS	Organization for the Advancement of Structured Information Standards
REST	REpresentational State Transfer
RPC	Remote Procedure Call
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
WCF	Windows Communication Foundation
WSDL	Web-Sevice Description Language
XML	Extensible Markup Language

Listings