Service-orientierte Architektur

und Web Services

Seminararbeit

in der Projektgruppe

Modulares Enterprise Architecture Management (EAM) System

Betreuerin: Dr. Ulrike Steffens

vorgelegt von: Roland Koppe
Osterkampsweg 120
26131 Oldenburg

E-Mail: roland.koppe@informatik.uni-oldenburg.de

Abgabetermin: Oktober 2007

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation service-orientierter Architektur

Das Bestehen am Markt erfordert von kleinen bis großen Unternehmen angebotene Produkte und Dienstleistungen möglichst schnell, günstig und damit auch effizient anzubieten. Unternehmen und Organisationen, die sich an den Trends des Marktes orientieren, benötigen die Fähigkeit zur schnellen Reaktion auf gestellte oder zukünftige Nachfragen. So kann beispielsweise ein verspäteter Markteintritt, also die verspätete Bereitstellung von Produkten oder Dienstleistungen, eine erhebliche Umsatzeinbuße bedeuten (vgl. Burghardt, M. (2006), S. 34f).

Die möglichst schnelle Reaktion auf den Markt und die damit verbundene notwendige Flexibilität, die ein Unternehmen daher besitzen muss, zeichnen sich auf die zu Grunde liegenden Softwaresysteme[1] ab. So ist zu erwarten, dass ein Softwaresystem, welches nur starr in einem bestimmten Rahmen von Anwendungen funktionsfähig ist und nur dort effizient arbeiten kann, nicht ohne größeren Aufwand auf andere beziehungsweise mögliche zukünftige Bedürfnisse angepasst werden kann.

Dieser Problematik kann mit einer service-orientierten Sichtweise begegnet werden. So können Geschäftsprozesse und Teile von Geschäftsprozessen auf einzelne Dienstleistungen (Services) abgebildet werden. Diese Dienstleistungen können in geeigneter Weise kombiniert und wieder verwendet werden, so dass eine relativ schnelle Umstrukturierung von Geschäftsprozessen möglich wird.

1.2 Ziele dieser Arbeit

Die service-orientierte Architektur bietet eine allgemeine und abstrakte Sichtweise auf die Organisation von Softwaresystemen. Diese Arbeit gibt einen Überblick der Grundprinzipien der service-orientierten Architektur.

Die konkrete Umsetzung einer service-orientierten Architektur kann prinzipiell durch verschiedene service-orientierten Technologien erfolgen. Diese Arbeit bietet einen Einblick in die Umsetzung einer service-orientierten Architektur unter Verwendung von Web Services.

[1] Die Bezeichnung Softwaresystem wird hier weitergehend auch als Synonym zum Begriff Informationssystem verstanden.

1.3 Aufbau der Arbeit

Dieser Einleitung folgt ein Kapitel der Grundlagen service-orientierter Architekturen.

Kapitel 3 zeigt die mit einer service-orientierte Architektur verbundene Identifikation von Softwaresystemen und Anwendungslandschaften. Es werden mögliche Regeln für die Umsetzung von Qualität in einer service-orientierten Architektur genannt.

Darauf aufbauend gibt Kapitel 4 einen Einblick in die Anwendung und Realisierung einer service-orientierten Architektur durch Web Services.

Der Abschluss bietet eine Zusammenfassung und einen Ausblick im Bereich von Web Services und service-orientierter Architektur.

2 Service-orientierte Architektur

Der Begriff der service-orientierten Architektur beschreibt ein ganzheitliches Konzept für Softwaresysteme. Die Funktionalität des Softwaresystems wird dabei durch miteinander in Beziehung stehende Komponenten realisiert. Komponenten sind hier als Services, also als Dienste zu sehen. Die Interaktion zwischen diesen Diensten erfolgt im Allgemeinen auf Basis von Nachrichtenaustausch.

Eine Definition der service-orientierten Architektur (kurz SOA) wird von Erl geliefert (Erl, T. (2005), S. 54).

> „SOA is a form of technology architecture that adheres to the principles of service-orientation. When realized through the Web services technology platform, SOA establishes the potential to support and promote these principles throughout the business and automation domains of an enterprise."

Dabei stellen Web Services nicht die einzige Möglichkeit dar um eine SOA zu realisieren. Auch andere Technologien wie beispielsweise CORBA, DCOM, WCF oder EJB können zum Einsatz kommen um eine service-orientierte Architektur zu realisieren (vgl. Wikipedia. (2007)).

2.1 Service-Orientierung

Die grundlegenden Prinzipien der Service-Orientierung können nach Erl (Erl, T. (2005), S. 37) festgehalten werden als:

- Lose Koppelung: Services haben nur minimale Abhängigkeiten untereinander und benötigen nur ein „Wissen" darüber, dass es auch noch andere Services gibt,
- Service Vertrag: Services haben eine Art Kommunikationsvereinbarung, die durch eine oder mehrere Beschreibungen festgelegt wird,
- Autonomie: Services haben die (alleinige) Kontrolle über die ihnen zugrunde liegende Logik,
- Abstraktion: Services geben nur ihre Beschreibung im Service Vertrag nach außen und verstecken ihre Umsetzung,
- Wiederverwendbarkeit: Die Logik befindet sich aufgeteilt in Services mit dem Ziel, Services mehrfach einsetzen zu können,

- Komponierbarkeit: Services können so koordiniert und zusammengefasst werden, dass neue zusammengesetzte Services entstehen,
- Zustandslosigkeit: Services minimieren die Speicherung von Zustandsinformationen für eine bestimmte Aktivität und
- Auffindbarkeit: Services sind äußerlich so beschrieben, dass sie gefunden und bewertet werden können.

Die genannten Prinzipien geben einen Eindruck der Möglichkeiten, die durch Service-Orientierung geschaffen werden können.

Bei der Umsetzung einer service-orientierten Architektur nach diesen Grundprinzipien scheinen Probleme wie beispielsweise der Austausch von Komponenten innerhalb von Softwaresystemen und die Erweiterung von Softwaresystemen durch Services durchaus machbar. Außerdem ist vorstellbar, dass bestehende Anwendungslogik in Services gekapselt werden kann, um so die Interoperabilität zu unterstützen beziehungsweise zu ermöglichen. Die Interoperabilität zeigt sich damit beispielsweise direkt bei der Integration von Softwaresystemen in einem Unternehmen (intra-enterprise) oder auch bei Unternehmensbeziehungen (cross-enterprise) (vgl. Krafzig, D.; Banke, K.; Slama, D. (2004), S. 89f).

2.2 Services

Die service-orientierte Architektur basiert auf der Verwendung und Nutzung von Services, also Diensten. Ein Dienst kann dabei einen beliebigen Ausschnitt eines Geschäftsprozesses und damit einen bestimmten Bereich der Anwendungslogik, die zur Realisierung des Geschäftsprozesses benötigt wird, abbilden.

Kapitel 3 geht auf die Qualität bei der Umsetzung einer service-orientierten Architektur ein und zeigt Richtlinien zur Kapselung von Anwendungslogik und Geschäftsprozessen in Services.

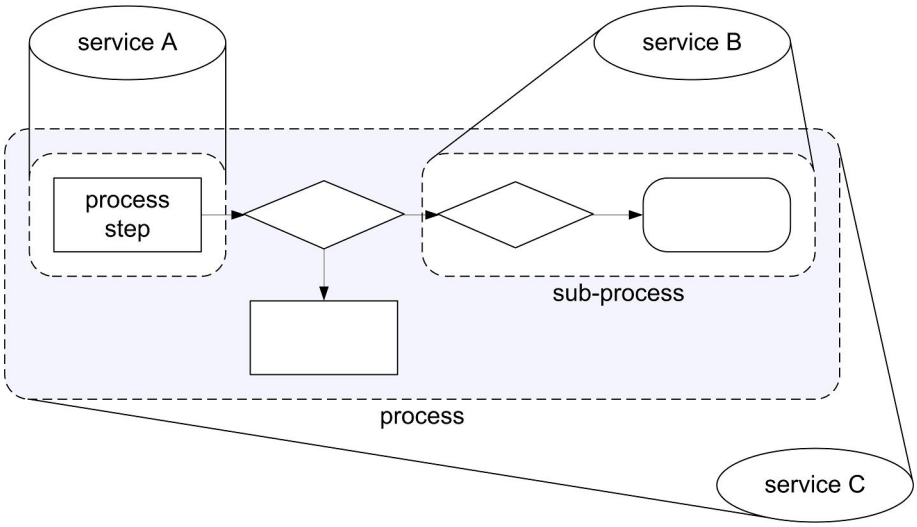

Abb. 2.1: Services als Teile eines Geschäftsprozesses[2]

Services können in beliebiger Weise kombiniert und verwendet werden (Abb. 2.1). Die Möglichkeiten zur Verwendung eines Services wird durch seine Beschreibung festgelegt. Der Austausch von Informationen zwischen Services erfolgt dann über das Senden und Empfangen von Nachrichten zwischen diesen Services.

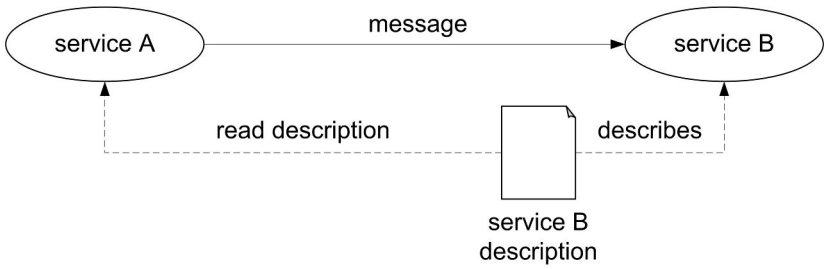

Abb. 2.2: Services, Beschreibung und Nachricht[3]

Abb. 2.2 zeigt zwei Services, die miteinander interagieren können. Diese Interaktion wird durch die Beschreibung des Services B ermöglicht. Service A liest dazu diese öffentlich zugängliche Beschreibung B. Diese Beschreibung gibt unter anderem Informationen zum Aufruf und zur Verwendung des Services B bekannt. Service A kann nun in einer geeigneten Form, die durch die Servicebeschreibung B festgelegt wurde, an B eine Nachricht senden, der diese dann verarbeiten kann.

Die Stärke der Kommunikation zwischen Services über Nachrichten liegt in der Unabhängigkeit dieser Services begründet. So ist diese Kommunikation beispielsweise plattformunabhängig. Weiterhin benötigen Services selbst kein Wissen über die technische Realisierung der Kommunikation (Hess, A.; Humm, B.; Voß, M. (2006)).

Abschnitt 4 geht auf die Umsetzung der Kommunikation mit Web Services ein.

---

[2] Quelle: eigene, vgl. Erl, T. (2005), S. 34
[3] Quelle: eigene, nach Erl, T. (2005), S. 36

3 Qualität in service-orientierten Architekturen

Der Einsatz und der Aufbau service-orientierter Architekturen zur Integration und zur Schaffung einheitlicher Softwaresysteme erfordert grundsätzlich eine hohe Qualität des Vorgehens zur Entwicklung solcher Architekturen. Anforderungen an entstehende Systeme können beispielsweise in der Zuverlässigkeit, der Effizienz und der Änderbarkeit gesehen werden.

Hess, A.; Humm, B.; Voß, M. (2006) geben in ihrem Artikel „Regeln für service-orientierte Architekturen hoher Qualität“ einen möglichen Leitfaden zur Entwicklung solcher service-orientierter Architekturen. Die Verwendbarkeit dieser Regeln wurde durch Praxiserfahrungen bei der Umsetzung großer Projekte bestätigt.

Im Folgenden werden diese Regeln kurz angesprochen. Vertiefungen sind in der aktuellen Literatur zum Thema SOA zu finden, vergleiche dazu auch das Literaturverzeichnis.

3.1 Anwendungslandschaft

Die Anwendungslandschaft von Unternehmen und Organisationen unterliegt generell einer bestimmten Ausprägung der Geschäftsarchitektur. Eine Geschäftsarchitektur basiert dabei auf einer Geschäftsstrategie, Geschäftsdomänen, Geschäftsprozessen, Geschäftsfunktionen und Geschäftsobjekten, die für eine Unternehmung als charakteristisch bezeichnet werden können.

Die konkrete Anwendungslandschaft einer Unternehmung basiert nun auf diesen Teilen der Geschäftsarchitektur. So besteht eine Anwendungslandschaft aus Anwendungsdomänen, Anwendungskomponenten, Anwendungsservices und Anwendungsservice-Operationen.

Zur Identifikation der Anwendungslandschaft wird eine Referenzarchitektur für Anwendungslandschaften vorgeschlagen. Abb. 3.1 zeigt diese Referenzarchitektur. So wird die Anwendungslandschaft vertikal in Anwendungsdomänen und horizontal in Servicekategorien unterteilt. Diese Aufteilung kann recht übersichtlich die Zugehörigkeit der einzelnen Komponenten einer Anwendungslandschaft aufzeigen.

Krafzig, D.; Banke, K.; Slama, D. (2004) empfehlen in Enterprise SOA – Service-Oriented Architecture Best Practices zur Beherrschung der Komplexität ebenfalls die Dekomposition von Business Use Cases (Anwendungsfällen, in etwa vergleichbar mit

Anwendungsdomänen) und Technical Layers, welche hier mit Servicekategorien vergleichbar sind (vgl. Krafzig, D.; Banke, K.; Slama, D. (2004), S. 286f und S. 82f).

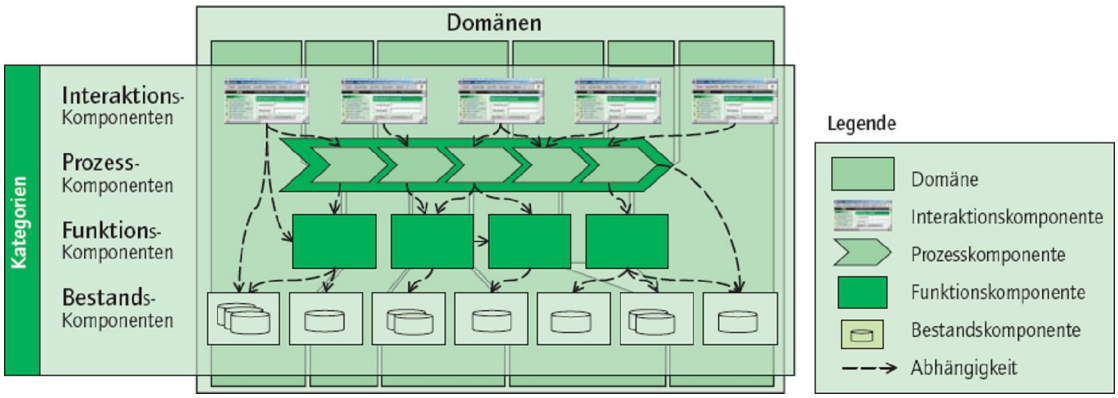

Abb. 3.1: Referenzarchitektur Anwendungslandschaft[4]

Dabei dienen Bestandskomponenten dem Zugriff und der Verwaltung von Datenbeständen. Funktionskomponenten stellen die fachliche Funktionalität einer Domäne bereit und unterstützen damit die Geschäftsfunktionen eines Unternehmens. Prozesskomponenten definieren und unterstützen Geschäftsprozesse, die als Workflows vorliegen können, und steuern damit Geschäftsfunktionen auf Basis bestimmter Geschäftsregeln. Schließlich bieten Interaktionskomponenten Benutzern die Möglichkeiten zur Interaktion und Kommunikation mit der Anwendungslandschaft. Das Idealbild einer Interaktionskomponente stellt für Benutzer eine einheitliche Sicht auf eine möglicherweise heterogene Anwendungslandschaft dar.

Die dargestellten Schichten der Anwendungslandschaft dienen hier der idealisierten Sicht auf die Trennung der Funktionalität einzelner Komponenten. Die einzelnen Schichten dieser Referenzarchitektur können hier durchaus alle Schichten beispielsweise einer Drei-Schichten-Architektur umsetzen. Eine Komponente aus dem Bereich der Bestandskomponenten kann also sowohl die Datenhaltung, Anwendungslogik und Präsentation beherbergen.

3.2 Regeln zur Schaffung hoher Qualität in service-orientierten Architekturen

Hess, A.; Humm, B.; Voß, M. (2006) geben in ihrem Artikel eine Reihe von Regeln an, die ihrer Praxiserfahrung nach, die Qualität service-orientierte Architekturen unterstützen. Diese Regeln finden sich auch in den Grundprinzipien der Service-Orientierung wieder. Abschnitt 2.1 zeigte diese Grundprinzipien nach Erl, T. (2005).

[4] Quelle: in Hess, A.; Humm, B.; Voß, M. (2006)

Es folgt hier ein Auszug dieser Regeln. Die Aufführung dieser Regeln kann als Katalog und Richtlinie für die Entwicklung von service-orientierten Architekturen gesehen werden. Detaillierte Informationen zu diesen Regeln und entsprechendem Vorgehen findet sich in der einschlägigen Literatur.

3.2.1 Regeln für fachliche Komponenten

- R1: Komponenten sollen nach fachlichen Kriterien gebildet werden.
- R2: Alle Operationen einer Komponente sollen genau aus einer Kategorie (Bestand, Funktion, Prozess, Interaktion) sein.
- R3: Komponenten unterschiedlicher Servicekategorien sollen ausschließlich Abhängigkeiten gemäß der Referenzarchitektur Anwendungslandschaft (Abb. 3.1[5]) haben.
- R4: Die Abhängigkeiten (kennt / ruft auf / bezieht Daten von) zwischen Komponenten sollen einen gerichteten azyklischen Graphen bilden.
- R5: Komponenten sollen so geschnitten werden, dass sie intern einen engen Zusammenhalt haben und untereinander gering gekoppelt sind.
- R6: Bestandskomponenten sollen die Datenhoheit über alle Geschäftobjekte haben.

3.2.2 Regeln für das Design von Services und ihren Operationen

Es wird zwischen verschiedenen Operationen, die ein Service anbietet, unterschieden. Elementare Operationen dienen zur Repräsentation einfacherer Funktionalität. Zusammengesetzte Operationen können einfache oder wiederum zusammengesetzte Operationen enthalten. Schließlich soll die Orchestrierbarkeit[6] es ermöglichen auf einfache Weise Geschäftsprozesse abzubilden.

- R7: Jede Serviceoperation soll ausschließlich fachliche Funktionalität bereitstellen und nichts über die eingesetzte Implementierungstechnik verraten.

---

[5] im Originaltext handelt es sich um Abb. 2

[6] Bei der Komposition von Services lassen sich Orchestrierung und Choreographie unterscheiden. Das World Wide Web Consortium (W3C) gibt in Haas, H; Brown, A. (2004) eine Definition dieser beiden Begriffe, im Speziellen für Web Services.

- R8: Die Parameter von Serviceoperationen (Eingabe- und Rückgabeparameter) sollen niemals Referenzen auf interne Objekte enthalten.

- R9: Die elementaren Operationen einer Komponente sollen in Normalform, das heißt vollständig und redundanzfrei sein.

- R10: Orchestrierbare Serviceoperationen sollen grobgranular sein.

- R11: Orchestrierbare Operationen sollen, falls fachlich sinnvoll und möglich, idempotent sein.[7]

- R12: Operationen von Funktions- und Bestandskomponenten sollen kein Wissen über den Kontext haben, in dem sie aufgerufen werden, und entsprechend über den Aufrufkontext keine Annahmen machen.
  Bei dem Aufrufkontext kann zwischen Sessionkontext, Transaktionskontext und Batch-/Online-Betrieb unterschieden werden.

Das Grundprinzip der losen Koppelung kann, den Autoren nach, an die Entfernung der einzelnen Komponenten angepasst werden. So zeigt Abb. 3.2 eine Einschätzung für Vertrauen, Wissen und Koppelung in Abhängigkeit von der Entfernung. Demnach wird zwischen den verschieden Koppelungsmechanismen (aufsteigend, beginnend mit eng gekoppelten Komponenten) Intraprozesskommunikation synchron, Request/Reply synchron, Message asynchron und Publish/Subscribe unterschieden.

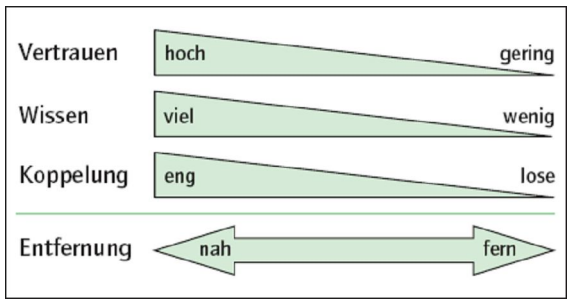

Abb. 3.2: Vertrauen, Wissen und Koppelung in Abhängigkeit der Entfernung[8]

Eine Übersicht enger und loser Koppelung kann auch Krafzig, D.; Banke, K.; Slama, D. (2004), S. 46ff entnommen werden.

- R13: Der Koppelungsmechanismus sich aufrufender Komponenten soll so gewählt werden, dass der Grad der Koppelung mit zunehmender Entfernung abnimmt.

---

[7] Die mehrmalige Ausführung einer Operation hat den selben Effekt, wie die einmalige Ausführung dieser Funktionen. Dieses gilt unter der Annahme, dass die an die Operation übergebenen Parameter auch die selben sind.

[8] Quelle: in Hess, A.; Humm, B.; Voß, M. (2006)

- R14: Aufrufe von Funktions- und Bestandsservices sollen stets in einem Transaktionskontext laufen. Aufrufe von Interaktions- und Prozessservices sollen in keinem Transaktionskontext laufen.

- R15: Je größer die Entfernung zweier sich aufrufender Komponenten ist, desto geringer sollen die Annahmen für ein gemeinsames Verständnis fachlicher Begriffe sein.

4 Web Services

Web Services können dazu genutzt werden, eine service-orientierte Architektur umzusetzen. Die etablierten Standards im Bereich der Web Services ermöglichen es, plattformunabhängig und verteilt, Dienste anzubieten und zu nutzen. Weitergehende Bemühung in Richtung von semantischen Web Services haben zum Ziel die automatische Auswahl und Nutzung von Web Services zu ermöglichen.

Im Folgenden werden kurz wesentliche Standards zu Web Services aufgezeigt.

4.1 Universal Description, Discovery and Integration (UDDI)

Mit UDDI definiert die Organization for the Advancement of Structured Information Standards (OASIS) einen plattformunabhängigen Verzeichnisdienst für Web Services und andere elektronische, sowie auch für nicht elektronische Dienste. Ein solcher Verzeichnisdienst ermöglicht das Anbieten und die Nachfrage von Web Services, etwa wie die „Gelben Seiten“.

Ein solcher Verzeichnisdienst kann wiederum selbst durch einen Web Service realisiert werden. Dabei werden Informationen über Anbieter, Implementierung und Metadaten von Web Services gespeichert und für Abfragen zur Verfügung gestellt.

Anbieter von Web Service können nun UDDI verwenden um ihre Dienstleistungen anzubieten. Auf der anderen Seite können Konsumenten UDDI benutzen, um nach angebotenen Dienstleistungen zu suchen, die ihren gestellten Anforderungen genügen. Die Metadaten dienen dabei als Einsatzbeschreibung eines Dienstes.

Weiterhin definiert OASIS mit der UDDI Spezifikation:

- die Verwendung des Simple Object Access Protocol (SOAP), um Informationen und damit Dienste von Anbietern veröffentlichen zu können beziehungsweise um Konsumenten die Möglichkeit zu geben nach Angeboten im Verzeichnis zu suchen,
- die Formate von Datenmodellen für das Verzeichnis und die SOAP Nachrichten per Extensible Markup Language Schema (XML Schema),
- die Web Service Description Language (WSDL) zur Definition des SOAP Application Programming Interface (API) und

- so genannte tModels (technical models), die zur Identifizierung und Kategorisierung von Diensten dienen.

4.2 Simple Object Access Protocol (SOAP)

SOAP dient als Standard zum Austausch für Nachrichten zwischen Web Services. Es basiert dabei auf der Extensible Markup Language (XML). Das W3C beschreibt in seiner Empfehlung zu SOAP 1.2 (Gudgin, M.; Hadley, M. (2007)), SOAP als leichtgewichtiges Protokoll, welches prinzipiell beliebige Transportprotokolle für die Kommunikation verwenden kann. Typischerweise kommt für die Kommunikation aber das Hypertext Transfer Protocol (HTTP) zum Einsatz.

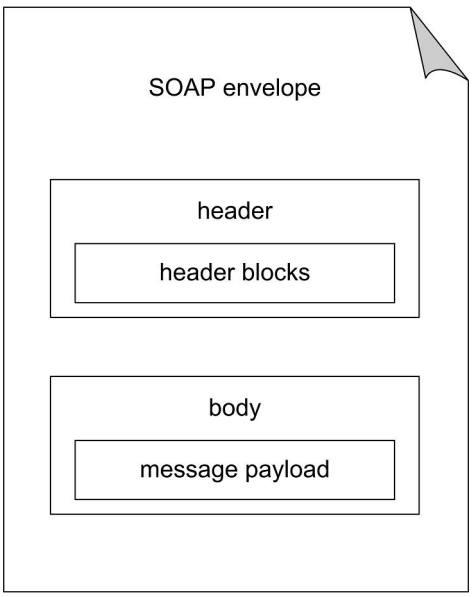

Abb. 4.1: Struktur einer SOAP Nachricht[9]

Zu den wichtigsten Zielen von SOAP können die Einfachheit und die Erweiterbarkeit des Protokolls gezählt werden. Abb. 4.1 zeigt die Struktur einer SOAP Nachricht. Dabei besteht eine solche Nachricht lediglich aus einer Hülle (envelope), einem Kopf (header) und der eigentlichen Nachricht (body).

Das Verfahren SOAP Messages with Attachments (Barton, J.; Thatte, S. et al. (2000)) bietet eine Möglichkeit an, neben denen in XML beschriebenen Daten beispielsweise auch binäre Daten als Anhang mitzugeben. Die Verpackung des Anhangs wird dabei mit dem MIME (Multipurpose Internet Mail Extensions) Verfahren realisiert.

[9] Quelle: eigene basierend auf Erl, T. (2005), S. 144

Bei der Nutzung von SOAP können im Wesentlichen zwei Arten der Kommunikation unterschieden werden.

- RPC-SOAP besitzt die Charakteristik eines vom Client ausgeführten entfernten Methodenaufrufs mit anschließender Bestätigung als Rückgabe vom Server. So kommt auch der Name RPC für Remote Procedure Call zustande.

- Message SOAP zeichnet sich im Gegensatz zu RPC-SOAP durch das „bloße“ senden einer Nachricht aus. Die Nutzlast (payload) der SOAP Nachricht kann hierbei ganze XML Dokumente aufnehmen und wird als „literal encoding“ bezeichnet.

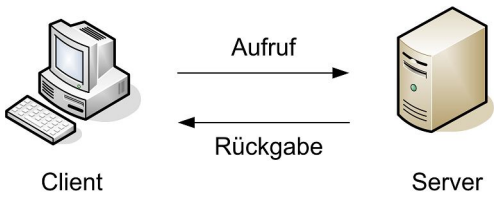

Abb. 4.2: SOAP Kommunikation: RPC-SOAP[10]

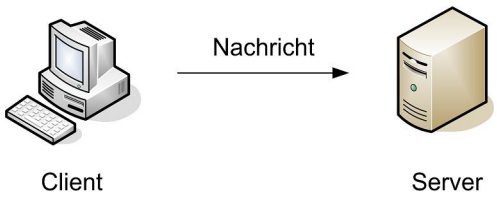

Abb. 4.3: SOAP Kommunikation: Message SOAP[11]

Eine synchrone Kommunikation kann also in der Verwendung von RPC-SOAP gesehen werden. Message SOAP hingegen stellt die Grundlage für eine asynchrone Kommunikation bereit. Weiterhin besteht die Möglichkeit einer Kombination beider Ansätze als dokumentenbasierter Aufruf, bei dem der Methodenaufruf über ein literal encoding realisiert wird (vgl. Sparrenberg, M. (2002), S. 15).

4.3 Web Service Description Language (WSDL)

Die Beschreibung der Schnittstellen eines Web Services kann mit Hilfe der Web Service Description Language (WSDL) vorgenommen werden. WSDL basiert dabei wiederum auf XML Ein WSDL Dokument liegt beim Anbieter eines Web Services bereit und ist durch eine bestimmte URL (uniform resource locator) zu erreichen. Im

[10] Quelle: eigene
[11] Quelle: eigene

WSDL Dokument benannte Endpunkte zeigen auf bestimmt Operationen eines Web Services (vgl. Christensen, E.; Curbera, F. (2001)).

Web Services können somit von Konsumenten genutzt werden, wenn diese sich zuvor über die Operationen eines Web Service informiert haben. Ein Konsumente eines Web Services fordert also zunächst das WSDL Dokument des Web Services an. Nach Erhalt des WSDL Dokuments kann der Konsument den angebotenen Web Service durch Kommunikationen mit SOAP verwenden.

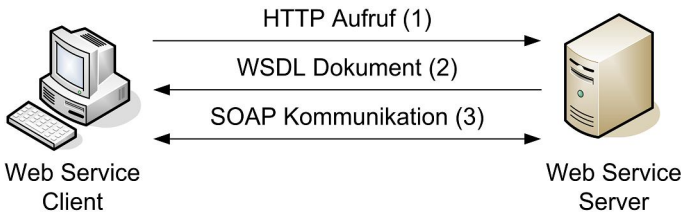

Abb. 4.4: Direkte Web Service Nutzung[12]

4.4 Web Service Architektur

Die Kombination der in den vorherigen Abschnitten angesprochenen Standards führt zu einer Web Service Architektur, die der Abb. 4.5 entspricht. Zunächst wird die Beschreibung eines Web Services veröffentlicht. Dann wird ein Web Service im Verzeichnis gesucht und ein Verweis auf dessen Beschreibung zurückgeliefert. Anschließend kann die Nutzung des Web Services erfolgen.

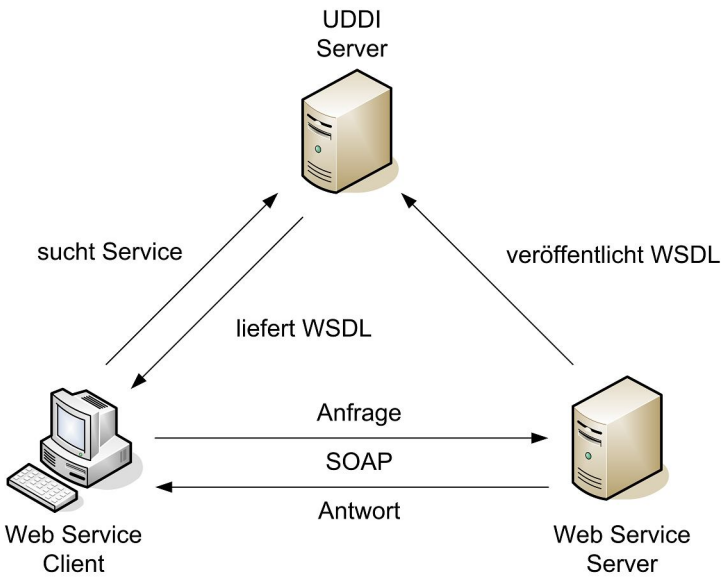

Abb. 4.5: Web Service Architektur

[12] Quelle: eigene

5 Zusammenfassung

Web Services stellen heute eine aktuelle Möglichkeit dar, um Dienstleistung in Form von möglicherweise über das Internet verteilten Diensten anzubieten. Web Services können aufgrund ihrer Architektur als plattformunabhängige und verteilte Komponenten von Softwaresystemen betrachtet werden.

Das übergeordnete Konzept der service-orientierten Architektur definiert Grundprinzipien für die Entwicklung und Nutzung solcher verteilter Strukturen. Die Ziele und Stärken einer service-orientierten Architektur können beispielsweise bei ihrer Flexibilität, der losen Koppelung, der Abstraktion und der Wiederverwendbarkeit von Services beobachtet werden.

Weiterhin können durch service-orientierte Architekturen Geschäftsprozesse von Unternehmungen abgebildet werden, indem Services auf Basis aktueller Anforderungen zu Unterstützung von Geschäftsprozessen kombiniert werden können.

5.1 Ausblick

Nach der Diskussion um Web Services sind recht aktuell Arbeiten um das so genannte Semantic Web und damit um semantische Web Services getreten. Wobei das automatische Auffinden von Web Services, das automatische Ausführen und die automatische Komposition sowie die Interoperabilität, beispielsweise in OWL-S[13], diskutiert werden.

Es ist zu erwarten, dass die Entwicklung von service-orientierten Architekturen weiter verfolgt und vertieft wird, um mögliche Potenziale beispielsweise im Bereich der Integration von Softwaresystemen weiter auszuschöpfen.

[13] Ontology Web Language for Services: Martin, D.; Burstein, M. (2004)

6 Literatur

Barton, J.; Thatte, S. et al. (2000): SOAP Messages with Attachments. http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211 (2007-10-21)

Burghardt, M. (2006): Projektmanagement – Leitfaden für die Planung, Überwachung und Steuerung von Entwicklungsprojekten. 7. Auflage, Publicis Corporate Publishing, Erlangen.

Christensen, E.; Curbera, F. (2001): Web Service Description Language. (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315 (2007-10-21)

Erl, T. (2005): Service-Oriented Architecture – Concepts, Technology, and Design. Prentice Hall, Upper Saddle River.

Frotscher, T.; Teufel, M. et al. (2007): Java Web Services mit Apache Axis2. entwickler.press, Frankfurt M.

Gudgin, M.; Hadley, M. (2007): SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). http://www.w3.org/TR/2007/REC-soap12-part1-20070427 (2007-10-21)

Haas, H; Brown, A. (2004): Web Services Glossary. http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211 (2007-10-21)

Hess, A.; Humm, B.; Voß, M. (2006): Regeln für service-orientierte Architekturen hoher Qualität. Informatik Spektrum Dezember 2006 Sonderdruck. http://www.sdm.de/web4archiv/objects/download/fachartikel/1/sdm_pub_voss_regelnfuersoa.pdf (2007-10-21)

Krafzig, D.; Banke, K.; Slama, D. (2004): Enterprise SOA – Service-Oriented Architecture Best Practices. Prentice Hall, Upper Saddle River.

Martin, D.; Burstein, M. (2004): OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122 (2007-10-21)

Sparrenberg, M. (2002): Austausch personenbezogener Daten zwischen Portalen mit Hilfe von Web-Service-Technologie. Diplomarbeit. Carl von Ossietzky Universität Oldenburg.

Wikipedia. (2007): Serviceorientierte Architektur. http://de.wikipedia.org/wiki/Serviceorientierte_Architektur (2007-10-21)

Anhang

A Werkzeuge für die Entwicklung von Web Services

Die folgenden Werkzeuge können bei der Entwicklung von Web Services in der Sprache Java unterstützen.

- Apache Tomcat[14] kann als Servlet Container für die Bereitstellung und Ausführung von Web Services in Java verwendet werden.
- Apache Axis2[15] ist eine freie Implementierung von SOAP und steht für die Sprache Java als auch für die Sprache C zur Verfügung.
- Eclipse Web Tools Platform (WTP) Project unterstützt komfortabel bei der Entwicklung Java basierter Web Services.
- Java2WSDL[16] wird als grafisches Werkzeug von dem Apache Axis2 Projekt in Form des Apache Axis2 Codegen Wizard bereitgestellt. Das Werkzeug erzeugt aus gegebenen Java Code eine WSDL Beschreibung.
- Service Archiver[17] wird auch von dem Apache Axis2 Projekt zur Verfügung gestellt. Das Werkzeug dient dazu, Web Service Dateien in ein Archiv zu speichern um auf einem Server mit Axis2 veröffentlichen zu können.
- SOAPMonitor als Teil des Apache Axis Projekts ermöglicht die Überwachung von SOAP Kommunikation zwischen Client und Web Services.
- TCPMon kann als ein Debug Werkzeug verwendet werden um die Kommunikation über TCP zu überwachen und beispielsweise SOAP Nachrichten zu versenden.

B Vorgehensmodelle

Als grundlegende Entwurfsentscheidung beim Vorgehen zur Entwicklung von Web Services können im Wesentlichen zwei Methoden unterschieden werden (Frotscher, T.;

[14] http://tomcat.apache.org/
[15] http://ws.apache.org/axis2/
[16] http://ws.apache.org/axis2/tools/index.html
[17] http://ws.apache.org/axis2/tools/index.html

Teufel, M. et al. (2007), S. 55ff; vgl. Krafzig, D.; Banke, K.; Slama, D. (2004), S. 164ff, Top-Down Approach und Bottom-Up Approach).

- Contract First bezeichnet einen Ansatz, bei dem zunächst die Entwicklung der Schnittstellendefinition, also die WSDL Beschreibung, für einen Web Service und danach die tatsächliche Implementierung der Anwendung auf Basis der WSDL Beschreibung erfolgt. Dieses Vorgehen hat den Vorteil ausschließlich in XML Schema definierte und standardisierte Datentypen verwenden zu können und damit die Interoperabilität eines Web Service zu sichern. Leider stehen für die Unterstützung dieses Ansatzes heute nur wenige Werkzeuge zur Verfügung.

- Code First steht im Gegensatz zum Contract First Ansatz. Beim Code First Ansatz wird zunächst die eigentliche Anwendung entwickelt und aus den dadurch entstandenen Klassen eine WSDL Beschreibung für den Web Service generiert. Dieses Vorgehen kann in größeren Integrationsprojekten allerdings als problematisch angesehen werden, da durch heterogene und meist getrennte Entwicklung von Service Anwendung und Client verschiedene Interpretationen von Daten und Datentypen zu erwarten sind. Entsprechend ist anzunehmen, dass darunter die Interoperabilität leiden kann. Es existieren verschiedene Werkzeuge zur Unterstützung des Code First Ansatzes.

Generell sollte, aus den bereits angesprochenen Gründen der Interoperabilität in größeren Systemen, unbedingt der Contract First Ansatz Verwendung finden. Die Aufwendungen zur Behebung von Problemen im Bereich der Interoperabilität durch heterogene Schnittstellen sind mit einer gemeinsamen und zuvor definierten Schnittstellendefinition beim Contract First Ansatz deutlich geringer einzuschätzen als beim Code First Ansatz.