Fachhochschule Offenburg

Studiengang Medien und Informationswesen

Web Services - Dienstorientierte Architekturen im Internet

Diplomarbeit

Bearbeiter: Hendrik Saly
Betreuer: Prof. Dr. Rüdebusch, FH Offenburg
Prof. Dr. Christian Roth, itelligence AG Berlin
Bearbeitungszeit: 01.10.2001 - 31.01.2002

FÜR MEINE FAMILIE.

Erklärung:
Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfaßt und nur unter Zuhilfenahme der angegebenen Quellen und Hilfsmittel angefertigt habe.

_____

Berlin, 28.01.2002

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Was sind Web Services?

Web Services (vgl. [Kre01]) sind eine Integrationstechnologie, welche die dynamische Kommunikation von Anwendungen über Plattform- und Sprachgrenzen hinweg ermöglicht. Dies wird durch zwei Eigenschaften erreicht: Zum einen sind Web Services durch die Verwendung von XML interoperabel und zum anderen erweitern sie die Architektur des Internets um eine dienstorientierte Infrastruktur. Ein einzelner Web Service repräsentiert die Schnittstelle einer solchen Anwendung, die über das Internet bereitgestellt wird.

Mit Hilfe von Web Services können bestehende Anwendungen über die bestehende Infrastruktur des Internets schnell und kostengünstig für andere Anwendungen verfügbar gemacht werden. Um diese Ziele realisieren zu können, baut die Web Service Architektur durchweg nur auf offenen und bereits etablierten Standards[1] auf. Web Services sind sowohl plattformunabhängig wie programmiersprachenunabhängig und bringen somit alle Voraussetzungen mit, Anwendungen in einem heterogenen Umfeld, wie beispielsweise das Internet, interagieren zu lassen. Mit der Einführung von Web Services vollzieht sich ein Paradigmenwechsel des Internets von einer dokumentenorientierten Architektur, hin zu einer service- oder dienstorientierten Architektur. Web Services sind eine Technologie, um sowohl Dienste über das Internet bereitzustellen, wie auch eine Abbildung von Geschäftsprozessen auf die Infrastruktur des Internets zu ermöglichen. Sie stellen Funktionen bereit, um Dienste zu beschreiben, zu veröffentlichen und zu finden. Sie besitzen einen Mechanismus für die Beschreibung ihrer eigenen Schnittstelle. So ist eine dynamische und flexible Interaktion mit ihnen möglich. Web Services lassen sich zu einer Art Prozessreihe zusammenschalten, indem ein Web Service einen anderen (oder mehrere) aufruft und die Ergebnisse auswertet. Sie lassen sich daher mit dem bereits bekannten Konzept des ”Softwareagenten” vergleichen, da sie über eine gewisse Intelligenz bzw. Kontextsensitivität verfügen. Darüber hinaus können sie aufgrund der dynamischen Schnittstellenbeschreibung spontan und in bestimmten Grenzen auch autonom in Aktion treten.

In der Welt der Web Services gibt es drei Rollen (vgl. [Kre01, S. 7 f.]):

- **den Dienstanbieter (Service Provider)**
  Technisch gesehen ist dies die Maschine, die den Web Service bereitstellt. Aus der Sicht von Unternehmen ist es der Eigentümer des Dienstes.
- **den Dienstnehmer (Service Requestor)**
  Aus technischer Sicht ist der Dienstnehmer eine andere Anwendung (z.B. ein anderer Web Service), die mit dem Dienst interagiert. Aus unternehmerischer Sicht ist es der Geschäftspartner, der den Dienst benötigt und nutzt.
- **das Dienstverzeichnis (Service Registry)**
  Das Dienstverzeichnis ist ein durchsuchbares Register, in dem sämtliche verfügbaren Web Services registriert und beschrieben sind. Hier haben die Dienstanbieter die Möglichkeit, ihre Services zu veröffentlichen. Sobald dies geschehen ist, kann ein Dienstnehmer den Service finden und nutzen.

Damit ein Web Service genutzt werden kann, müssen drei Phasen durchlaufen werden:

1. **Veröffentlichen (publish)**
   Um auf den Dienst zugreifen zu können, muss er vom Dienstanbieter veröffentlicht werden. Hierzu können Verzeichnisse oder Point-to-Point Protokolle (z.B. E-Mail) benutzt werden. Dies hängt von der Art des Dienstes und seiner Zielgruppe ab.
2. **Finden (find)**
   Nachdem der Dienst veröffentlicht wurde, kann er vom Dienstnehmer gefunden werden. Entweder er

[1] Hierzu zählen unter anderem das Hypertext Transfer Protokoll (HTTP), die Extensible Markup Language (XML), das Simple Object Access Protokoll (SOAP), die Web Services Description Language (WSDL) und Universal Description, Discovery and Integration (UDDI).

erhält direkt die Beschreibung des Dienstes (Service Description) oder er sucht in einem Verzeichnis nach dem geeigneten Dienst.

3. **Binden (bind)**
   Ist der betreffende Dienst gefunden worden, kann sich der Dienstnehmer nun mit Hilfe der Dienstbeschreibung an diesen binden. Er tritt auf Grundlage der Dienstbeschreibung nun mit dem Dienst in Kontakt indem er ihn aufruft.

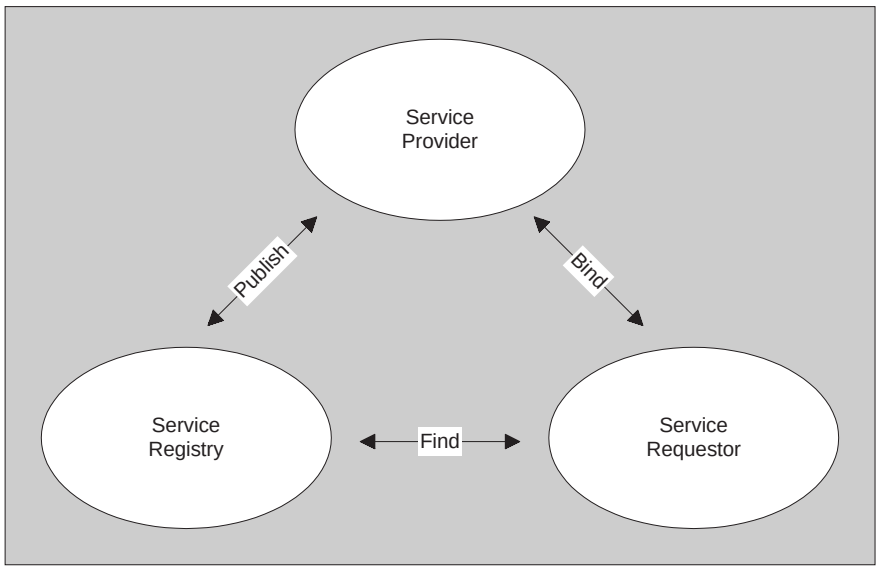

Abbildung 1: Rollen und Operationen in der Web Service Architektur (vgl. [Kre01])

1.2 Erläuterung und Begründung der Themenwahl

Web Services sind zur Zeit ”der Hype”. Die Motivation des Autors für diese Arbeit besteht darin, den tatsächlichen Nutzwert dieser neuen Technologie zu ergründen. Der Autor möchte zeigen, was zum heutigen Zeitpunkt möglich ist, welches Potential in der Technologie steckt und wo die Grenzen liegen. Die Arbeit soll ein objektiver Beitrag in der Diskussion um Nutzenwert, Potential und ”Real-World”-Tauglichkeit von Web Services sein. Das Thema wurde jedoch auch gewählt, weil Web Services eine Evolution in der Nutzung des Internets darstellen und dieser Schritt, der Meinung des Autors nach, möglicherweise entscheidend für die Zukunft des E-Business und der B2B Kommunikation sein wird.

1.3 Ziele der Arbeit

In dem für die Diplomarbeit angesetzten Bearbeitungszeitraum von vier Monaten soll das Konzept der Web Services analysiert und prototypisch umgesetzt werden. Technische Grundlagen von Web Services sollen dabei vor dem Hintergrund der Einsetzbarkeit in realen Anwendungen betrachtet werden. Die fertige Arbeit soll bei der Entscheidung helfen, eine serviceorientierte Architektur in einem Unternehmen für in- oder externe Prozesse einzuführen. Des weiteren kann die Arbeit als Leitfaden für die Entwicklung von Web Services verstanden werden. Allgemeine Aspekte dieser Entwicklung werden anhand eines konkreten Beispiels besprochen.

1.4 Thematische Abgrenzung

Diese Arbeit ist technischer Natur. Daher wird auf wirtschaftliche Faktoren nur am Rande eingegangen. Der Fokus liegt auf den Architekturen und Produkten von IBM, sowie auf der Programmiersprache Java bzw. dem J2EE Standard. Die .NET Architektur der Firma Microsoft wird nur am Rande betrachtet. Dies begründet sich vor allen Dingen darin, dass .NET ein völlig neues Paradigma, das weit über die Entwicklung von Web Services hinausgeht, darstellt und dies nicht Gegenstand dieser Arbeit ist. Es soll soweit wie möglich freie Software eingesetzt werden. Trotzdem wird für die clientseitige Implementierung das Microsoft SOAP Toolkit sowie die Programmiersprache Visual Basic verwendet. Dies soll die Interoperabilität der Web Service Technologie demonstrieren. Des weiteren werden die technischen Grundlagen des Internets nicht behandelt und als bekannt vorausgesetzt. Hierzu sei auf die zahlreich existierende Literatur und die Spezifikationen der Standardisierungsgremien verwiesen.

2 Motivation für Web Services

Das folgende Kapitel erläutert, aus welchen Gründen die Web Service Architektur entwickelt wurde. Es werden sowohl wirtschaftliche wie technische Faktoren betrachtet. Das Kapitel schließt mit einer Diskussion über die Motivationsgründe und deren derzeitigen praktischen Bezug.

2.1 Wirtschaftliche Faktoren

In diesem Kapitel werden die wirtschaftlichen Motivationsgründe für die Web Service Technologie behandelt. Welche Vorteile bringen Web Services, wenn sich ein Unternehmen dazu entscheidet, sie einzusetzen? Unternehmen sollten Web Services in zweierlei Hinsicht wahrnehmen: Zum einen als Integrationstechnologie und zum anderen als Kommunikationsgrundlage für Business-to-Business (B2B) Aktivitäten.

”Die größte Herausforderung für Unternehmen ist es heute, ihre heterogenen bestehenden Systeme und Applikationen miteinander so zu verbinden, daß die IT Systeme intern sowie auch extern mit Partnersystemen kommunizieren können.”[Shi01]

Es stellen sich heute bei der Kommunikation zwischen Anwendungen folgende Probleme: Wie in Kapitel 3.2.2 gezeigt wird, besteht Integrationsbedarf zwischen Anwendungen. Diese laufen auf verschiedenen Plattformen und sind in unterschiedlichen Programmiersprachen geschrieben. Ferner sind sämtliche Protokolle (z.B. IIOP, RMI) an ein Objekt-Modell gebunden und somit nicht universal einsetzbar. Aber genau dies ist die Voraussetzungen für weltweit verteilte Systeme, die Geschäftsprozesse abbilden sollen.

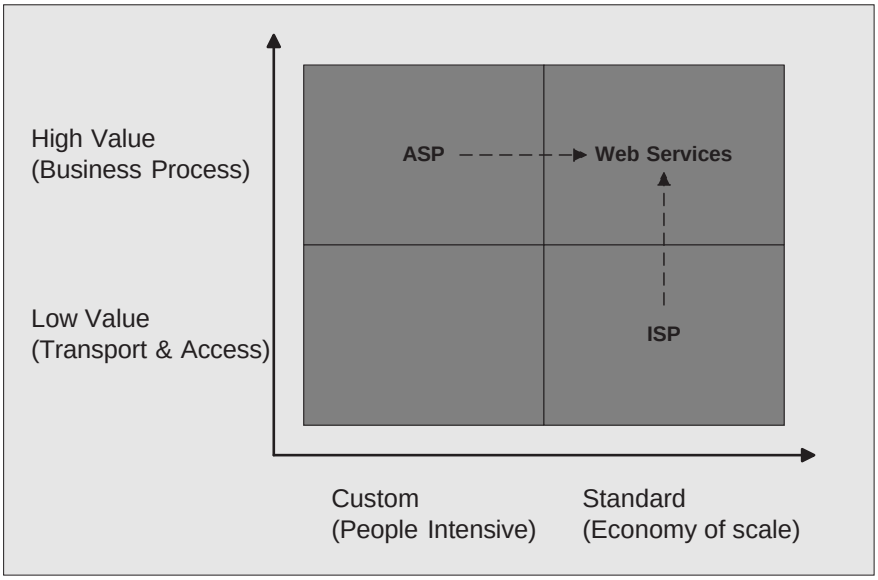

Abbildung 2: Web Services im Kontext von ASP (vgl. [Rol01])

IBM bezeichnet Web Services als ”dynamic e-business”. Dynamisch bedeutet, dass eine spontane Interaktion von Geschäftspartnern (oder technisch gesehen von Geschäftsprozessen) möglich ist. Dies wird vor allem durch das ”publish, find, bind” Konzept realisiert.

Markttransparenz
Nach [See01] ist die Vision der Web Services, aus unternehmerische Sicht, eine völlige, weltweite Markttransparenz. Durch die Eigenschaft, dass sich die Dienste selbst beschreiben, ist es denkbar, Softwareagenten darauf anzusetzen, den aktuellen Markt zu evaluieren. Dem zu Gute kommt auch die zentral zugängliche Datenhaltung

der registrierten Dienste. Dies ermöglicht, alle dort angemeldeten Unternehmen mit einzubeziehen.

Als Beispiel sei hier der Preisvergleich genannt. Es ist natürlich schon früher möglich gewesen, sich Preise automatisch vergleichen zu lassen. Das funktionierte aber nur bei relativ homogenen Gütern. Wurden die Preisanfragen komplexer oder die Güter uneinheitlich, war es nahezu unmöglich. Ein Agent (Web Service) kann wiederum andere Agenten anstoßen und so einen Prozess abbilden. Diese durchsuchen dann z.B. systematisch alle Registrierungsdatenbanken und vergleichen dort die Preise (und bei Bedarf weitere Attribute wie Lieferbedingungen oder Verfügbarkeit) des gewünschten Artikels. Das funktioniert nicht nur bei Produkten, sondern ebenso bei Dienstleistungen oder öffentlichen Diensten (E-Government[2]).

Wiederverwendbarkeit von Software
Hat eine Softwarekomponente eine definierte Schnittstelle, so ist es möglich, die Komponente wiederzuverwenden. Im Fall der Web Services ist das die SOAP-Schnittstelle, die alle Services besitzen. Web Services sind ”eine globale Integrationsplattform für Komponenten verschiedener Modelle” [Pro01, S. 7] und tragen so zur Wiederverwendbarkeit (sogar über Plattform- und Sprachgrenzen hinweg) bei.

Kosteneinsparung
Durch eine vollständige elektronische Abwicklung der Geschäftsprozesse, können Kosten reduziert werden. Die Web Service Technologie kann helfen, solche Prozesse zu ermöglichen und zu optimieren (z.B. im Supply Chain Management).

Einige Geschäftsprozesse können unter Umständen auch aus dem Unternehmen ausgegliedert und von anderen Dienstleistern übernommen werden. Web Services ermöglichen somit das Outsourcing von Prozessen. Da Web Services standardisierte Nachrichtenformate und selbstbeschreibende Schnittstellen besitzen, kann man sie als eine Art Softwarekomponente sehen, deren Nutzung eingekauft werden kann. Oft ist der Zukauf billiger, als die eigene Entwicklung. Dies ist vergleichbar mit dem Application Service Providing (ASP).

Die Gartner Group gibt folgende Einschätzung ab:

”By 2005, the aggressive use of Web services will drive a 30 percent increase in the efficiency of IT development projects, according to Gartner research presented this week at Gartner Symposium ITxpo 2001.”[PA01]

Die aggressive Verwendung der Web Service Technologie, wird im Jahr 2005 dazu führen, dass die Effizienz von IT Projekten um 30 Prozent ansteigen wird.

Time to market
Bereits bestehenden Applikationen eine Web Service Schnittstelle zu geben, ist ein relativ leichtgewichtiger Vorgang. Der Entwicklungsprozess, der in Kapitel 4.3.1 näher erläutert wird, ist vergleichsweise kurz und unproblematisch. Eine kurze ”Time to Market” wird auch dadurch erreicht, dass die verwendeten Teiltechnologien bereits bekannt und etabliert sind. Es existieren insbesondere für XML und Java Entwicklungswerkzeuge, die ”Rapid Application Development” (RAD) unterstützen. Mit dem Visual Studio ist RAD auch in der Microsoftwelt möglich.

2.2 Technische Faktoren

Neben den wirtschaftlichen Faktoren, existieren auch eine Reihe technisch motivierter Gründe, weshalb die Web Service Technologie sinnvoll ist.

[2] Die Anbindung von Diensten, die von Behörden angeboten werden. Als Beispiel sei hier Wählen über das Internet genannt.

2.2.1 Offene Standards

Eines der Ziele der Web Service Technologie ist die Integration von Anwendungen (vgl. 2.2.2) in ein heterogenes Umfeld. Dabei ergibt sich ein Kernproblem: Die Anwendungen müssen über unterschiedliche Plattformen und Programmiersprachen hinweg miteinander kommunizieren können. Sie müssen in jeder Hinsicht interoperabel sein. Diesem Anspruch werden die Web Services hauptsächlich durch die Verwendung von XML als universelles Nachrichtenaustauschformat gerecht. XML wird mittels SOAP übertragen. Beide erfüllen die Anforderung nach Interoperabilität.

Die Notwendigkeit offener Standards
Ohne offene Standards wäre das Internet nicht das, was es heute ist. Zunächst soll versucht werden, zu definieren, was ”offener Standard” bedeutet. Die IETF[3] ist neben dem W3C (WWW Consortium) eine der Einrichtungen, die die Internetstandards entwickelt und definiert. Unter anderem sind dies z.B. HTTP, POP, SMTP und NNTP. Diese Standards sind aus mehreren Gründen offene Standards:

- Die Spezifikationen sind für jedermann frei zugänglich.
- Es können von jedermann Änderungsvorschläge eingebracht werden.
- Die Verwendung ist kostenlos und lizenzrechtlich unbedenklich.

Im Gegensatz dazu stehen die Standards, Protokolle oder Datenformate, die nicht der Öffentlichkeit zugänglich sind. Dies sind dann proprietäre Lösungen, die von einem Unternehmen entwickelt werden. Beispiele hierfür sind das Dateiformat von Microsoft Word[4], das Net8 Protokoll von Oracle oder das Dateiformat von Adobe Photoshop. Diese führen natürlich zu Problemen beim Datenaustausch und zu einer Abhängigkeit gegenüber dem Hersteller. Weiterhin besteht die Gefahr, dass die Formate in mehreren Jahren nicht mehr lesbar sind, da sie nicht mehr weiterentwickelt oder von den Herstellern aufgegeben werden. Insbesondere rein binäre Formate werden dann völlig unbrauchbar.

Um weltweite Interoperabilität zu ermöglichen und diese möglichst langfristig zu sichern, muss ausschließlich auf offene Standards gesetzt werden. Hier spielt zur Zeit vor allen Dingen XML eine entscheidende Rolle, auf die im nächsten Kapitel näher eingegangen werden soll.

Die besondere Rolle von XML
XML wird im Kapitel 4.4.2 detailliert behandelt. Hier wird auf die besondere Rolle von XML in Bezug auf Interoperabilität eingegangen.

Keine Technologie war in den letzen drei Jahren so präsent und erfolgreich wie XML. Zuerst war man der Meinung XML würde irgendwann einmal HTML ersetzen (in Verbindung mit XSL). Dann wurde relativ schnell das tatsächliche Potential dieser Sprache erkannt. Seitdem ist XML in den unterschiedlichsten Aufgabengebieten im Einsatz. Diese sind unter anderem:

- Universelles, standardisiertes, offenes, menschenlesbares Textformat zur Strukturierung von Daten.
- Interoperables Austauschformat für Daten.
- Format zur Darstellung internationaler Alphabete (Unicode).

[3] Internet Engineering Task Force (www.ietf.org)
Die IETF ist eine internationale offene Gemeinschaft von Personen, die an der allgemeinen Entwicklung des Internets interessiert sind. Die Kommunikation findet hauptsächlich über die zahlreichen Mailinglisten in Kongressen statt. Jeder, der sich an Diskussionen beteiligen möchte, kann dies ohne Einschränkungen tun und ist somit auch automatisch Mitglied. Alle Standards der IETF werden in sogenannten RFCs (request for comments) festgehalten. Diese sind in einem offenen Datenformat (ASCII, Postscript) für jedermann einsehbar.

[4] Dateiendung .doc

- Universeller RPC Mechanismus (XML-RPC, SOAP).
- Nativer Datenspeicher für XML-Datenbankmanagementsysteme[5].
- Multichannel Anwendungen[6].
- Einheitliches Format für Konfigurationsdateien.

Durch das Textformat (Unicode) wird Migrationsfähigkeit und Langlebigkeit garantiert. XML überwindet das Problem der Heterogenität von Systemen auf drei Ebenen (vgl. [Jec01b]):

1. **Behebung der syntaktischen Heterogenität durch den Einsatz von XSD und XMI**
   XSD ermöglicht die Kommunikation auf der Basis eines gemeinsamen Wortschatzes und einheitlich definierter Begriffe. Weiterhin wird eine formale Beschreibung gewährleistet, was eine maschinelle Verarbeitung gestattet.
2. **Behebung der semantischen Heterogenität durch XSLT**
3. **Behebung der kommunikations-Heterogenität durch Verwendung von SOAP**

2.2.2 Interoperabilität

Die Interoperabilität ist eine Kerneigenschaft von Web Services. Erst die Fähigkeit, Anwendungen über alle Grenzen hinweg miteinander kommunizieren lassen zu können, macht aus den Web Services eine Integrationstechnologie.

Integration heterogener Systeme
Web Services sind eine netzwerkübergreifende Integrationstechnologie, die verschiedene Systeme und Anwendungen auf der Grundlage von Standard Protokollen und Technologien (Internet, SOAP) verbinden möchte. Dabei bildet das Internet als bestehende, überall verfügbare und bewährte Netz-Infrastruktur die Grundlage für die Kommunikation. Dies könnte die Schwierigkeiten bei der Integration heterogener Systeme stark verringern oder sogar beheben. Ein mögliches Szenario ist zum Beispiel die Nutzung von DCOM Komponenten der Windows-Welt aus einer J2EE Anwendung heraus, die auf einem UNIX-System läuft. Dies wäre vor allen Dingen im Geschäftsumfeld ein wichtiger Fortschritt, da die Teilnehmer eines Geschäftsprozesses (Lieferanten, Lagerhaltung, Zwischenhandel, usw.) in der Regel alle unterschiedliche Software auf unterschiedlicher Hardware mit unterschiedlichen Datenformaten nutzen.

Anbindung von Legacy Systemen
Unter dem Begriff Legacy[7]-System wird ein in sich geschlossenes Alt-System verstanden. Dies können sowohl ERP (Enterprise Resource Planning) Systeme, alte Mainframe Anwendungen oder Datenbanksysteme ohne gängige Schnittstelle sein.

Diese Systeme sind robust und haben sich über Jahrzehnte hinweg bewährt. Es ist in den meisten Fällen unmöglich, diese Systeme durch moderne zu ersetzen. Eine solche Umstellung wäre nicht nur technisch sehr problematisch, sondern würde sich auch betriebswirtschaftlich nicht rechnen. Es besteht vom funktionellen Aspekt her auch keine Notwendigkeit diese Systeme auszutauschen, da sie den gestellten Anforderungen gerecht werden. Aber diese Systeme müssen an moderne Systeme und an das Internet angebunden werden. Dabei geht es nicht nur um die Darstellung bestimmter Daten auf einer Webseite, sondern vielmehr darum, den Systemen die Kommunikation über Standard Protokolle zu ermöglichen. Weiterhin ist eine Unterstützung von XML sehr wichtig, um im heterogenen Umfeld des Internets Daten austauschen zu können.

[5] z.B. Tamino von der Software AG, http://www.softwareag.com
[6] Realisierung endgeräteunabhängiger Informationscodierung (z.B. über XSLT)
[7] eng.: Vermächtnis

Die Anbindung von Legacy Systemen wird oft auch mit dem Begriff Enterprise Application Integration (EAI) bezeichnet. Prinzipiell wird zwischen drei Integrationsstufen unterschieden (vgl. [Jec01a]):

- **Datenintegration**
  Austausch von Daten zwischen einzelnen Systemen. Diese Aufgabe kann z.B. XML erfüllen, da ein gemeinsamer Wortschatz und formale Strukturierungsregeln festgelegt werden können. Außerdem haben XML Daten eine semantische Bedeutung.
- **Funktionsintegration**
  Funktionsintegration wird durch RPC/RMI Mechanismen erreicht. Dies kann z.B. mittels SOAP geschehen.
- **Prozessintegration**
  Um eine Prozessintegration zu ermöglichen, müssen sich Dienste publizieren und auffinden lassen, sowie dynamisch rekonfigurierbar sein. Diese Aufgaben können z.B. von UDDI und WSDL übernommen werden.

Auf allen Integrationsstufen können Web Services (oder Teile davon) eingesetzt werden. Doch wie sieht eine solche Anbindung konkret aus?

Ein Lösungsvorschlag, um solche Systeme zu öffnen und ihre Dienste im Netz nutzbar zu machen, ist die Kombination von J2EE und Web Services.

Innerhalb der J2EE Architektur von SUN existiert die sogenannte ”J2EE Connector Architecture”[8] mit deren Hilfe es möglich ist, Legacy Systeme mit einer J2EE basierten Anwendung zu verbinden. Falls keine zusätzliche Geschäftslogik erforderlich ist, kann direkt vom ”Front Controller” über die Connector Architecture auf das Alt-System zugegriffen werden. Ist weitere Geschäftslogik notwendig, findet der Zugriff über ein EJB-Tier statt. Nun muss ein Web Service View geschaffen werden und die Anbindung ist vollzogen. In der Vision der Web Services soll so etwas dann automatisch geschehen. In der Tat unterstützt die aktuelle Version 6.1 des Applikationservers Weblogic von Bea die Funktion direkt aus EJB’s vollautomatisch Web Services zu generieren (vgl. [BEA01]).

Weitere Technologien für die Anbindung von Mainframerechner sind z.B. der IONA Mainframe Integrator oder IBM MQSeries.

Abbildung von Geschäftsprozessen
Durch die Möglichkeit der Prozessintegration mittels Web Services, lassen sich mit dieser Technologie auch netzwerkübergreifende Geschäftsprozesse realisieren.

Anwendungen und Prozesse werden immer komplexer und eine Gesamtanwendung setzt sich immer öfter aus verteilten Komponenten zusammen, die sich gegenseitig verstehen müssen. Dabei ist Interoperabilität von besonderer Bedeutung. Web Services sind geeignet, solche Aufgaben zu lösen. Sie können durch die Fähigkeit spontan miteinander zu agieren einen Workflow abbilden. Ein solcher Workflow oder Prozess wird durch die WSFL (Web Services Flow Language) spezifiziert. WSFL ist eine auf XML basierende Sprache, die verschiedene Web Services nacheinander aufruft, wobei das Resultat eines Services als Eingabe für den nächsten verwendet werden kann. Es gibt Kontrollstrukturen, die in Abhängigkeit von bestimmten Ereignissen den Ablauf festlegen.

Beispiel für einen Geschäftsprozess:

Ein Geschäftsreisender, der viel mit dem Flugzeug unterwegs ist, möchte nicht nur seine Flüge buchen, sondern auch noch automatisch seine Vielfliegerprogramme möglichst effizient ausnutzen. Er möchte automatisch seine Hotels buchen lassen. Dort möchte er jeweils einen Tisch für das Abendessen reservieren lassen. Falls er ab und zu mit der Eisenbahn fahren muss, soll der Fahrschein am Hotel für ihn bereitliegen. Weiterhin möchte er, dass die Rechnung direkt vom Firmenkonto abgebucht wird, er aber immer noch zu jeder Zeit und an jedem

[8] http://java.sun.com/j2ee/connector/

Ort einen Überblick über seine ganzen Reisevorgänge und deren Kosten hat. Dieses System sollte so flexibel sein, dass bei Flugausfällen bzw. ausgebuchten Hotels entsprechend umdisponiert wird. So ein System ist mit derzeitig verfügbaren Technologien kaum zu realisieren. Die Problematik liegt in den heterogenen Systemen der einzelnen Parteien, die nicht miteinander kommunizieren können. Es fehlt an standardisierten Schnittstellen für eine Maschine-Maschine Kommunikation. Für die großen Unternehmen gibt es zwar Lösungen wie EDIFACT, aber für kleinere Betriebe wie z.B. Mittelklasse Hotels ist dies nicht bezahlbar. Ferner benötigen diese Lösungen eine bestimmte Infrastruktur, wohingegen Web Services auf dem Internet aufsetzen. Dies ist überall verfügbar und bezahlbar. Die folgende Abbildung stellt die einzelnen Geschäftsprozesse für das oben eingeführte Beispiel dar:

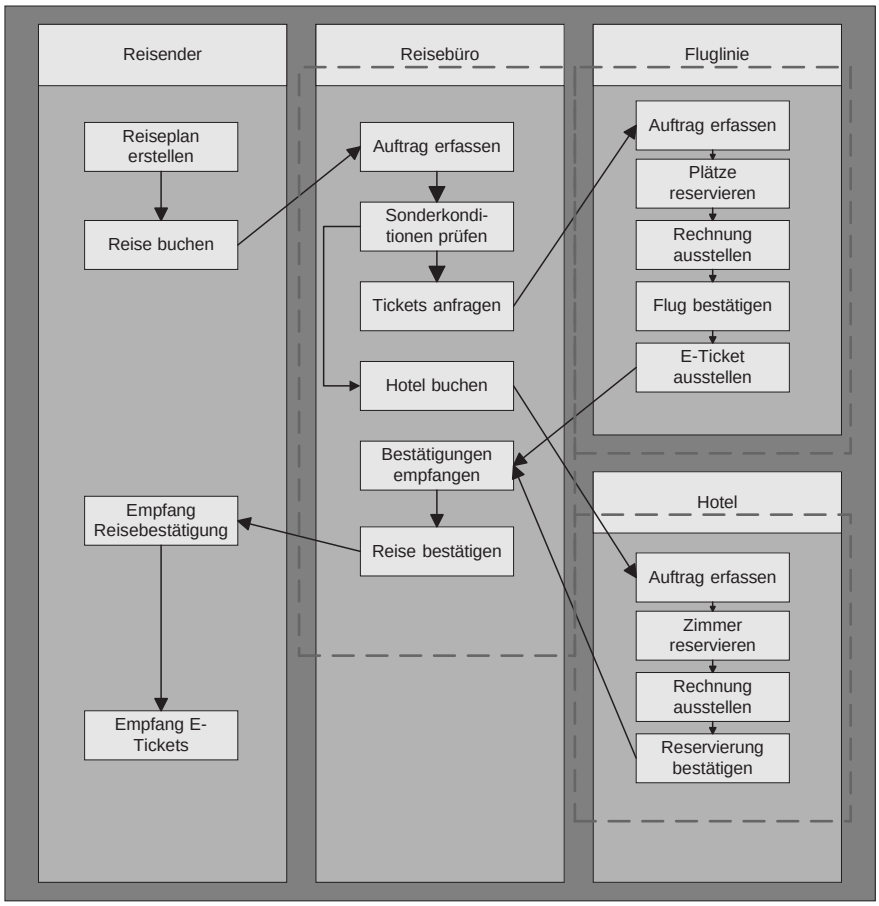

Abbildung 3: Abbildung eines Geschäftsprozesses

2.2.3 Formale Beschreibung

Erst die Eigenschaft der Web Services, sich selbst zu beschreiben, ermöglich die angestrebte automatisierte Kommunikation zwischen Anwendungen via Internet. Doch die Beschreibung eines Dienstes, muss hierfür nach einer genau festgelegten Syntax erfolgen. Sie muss formal sein, damit Maschinen respektive Anwendungen untereinander kommunizieren können. Diese Voraussetzung wird von der bisher existierenden Infrastruktur des Internets nicht erfüllt. Darüber hinaus liegt nicht nur eine formale Beschreibung der Schnittstelle vor, sondern auch die Nutzdaten sind formal beschrieben. Die wird durch das SOAP Protokoll sichergestellt.

2.3 Diskussion

Ob die Web Services das halten was sie versprechen, kann im Moment schlecht beurteilt werden. Aber da vor allem große Konzerne wie IBM, Microsoft, Sun, Orcale und auch Hewlett Packard in die Entwicklung involviert sind und sie mit enormer Geschwindigkeit vorantreiben, kann dies als Indiz gesehen werden, dass die Web Services wirklich eine ganze Reihe Probleme lösen können. Gerade Teiltechnologien wie z.B. UDDI oder ebXML haben in den letzten Monaten enormen Zulauf erfahren. Dass Integration eines der zentralen Themen heute und insbesondere der Zukunft ist, steht außer Frage. Web Services können hier einen wichtigen Beitrag leisten. Die Idee, das Internet für alle Kommunikationsaufgaben zu nutzen, wird schon allein deshalb erfolgreich sein, da es eine stabile und überall verfügbare[9] Infrastruktur darstellt, die in den letzen Jahren immer wichtiger geworden und aus dem Alltag nicht mehr wegzudenken ist.

[9] eng: pervasive

3 Aktueller Stand

In diesem Kapitel wird der aktuelle Stand auf dem sich die Web Service Technologie befindet untersucht. Zunächst wird versucht, eine Definition für den Begriff ”Web Service” zu finden. Danach werden die Konzepte von verteilten Systemen besprochen, um später die Web Service Technologie einordnen zu können. Ferner werden die wichtigsten Spezifikationen und einige Produkte vorgestellt. Schließlich folgt ein allgemeiner Überblick über den allgemeinen Stand der Technologie und über bereits verfügbare Web Services. Eine Diskussion möglicher Probleme schließt das Kapitel ab.

3.1 Definition von Web Services

IBM definiert Web Services folgendermaßen:
”A Web service is an interface that describes a collection of operations that are networkaccessible through standardized XML messaging. A Web service is described using a standard, formal XML notion, called its service description. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web Services fulfill a specific task or a set of tasks. They can be used alone or with other Web Services to carry out a complex aggregation or a business transaction.” [Kre01, S. 6]

IBM sieht Web Services also als eine Schnittstelle, die eine Sammlung bestimmter über das Netzwerk erreichbarer Operationen beschreibt. Die Kommunikation findet mittels standardisierten XML Nachrichten statt. Ein Web Service wird durch seine Dienstbeschreibung definiert, die ebenfalls als XML Dokument vorliegt. In dieser Dienstbeschreibung finden sich alle notwendigen Informationen, um den Dienst zu nutzen. Dies sind vor allem das genaue Nachrichtenformat, Transportprotokolle und die Adresse, unter welcher der Dienst zu erreichen ist. Die Schnittstelle verbirgt die Implementierungsdetails und macht den Service so plattform- und programmiersprachenunabhängig. Dies erlaubt Web Service basierten Anwendungen entkoppelt[10] von anderen Web Services, komponentenorientiert und unter Verwendung von Cross-Technologien[11] implementiert zu werden. Web Services können sowohl als alleinstehende Anwendung genutzt werden, aber es ist auch möglich durch gezieltes Zusammenschalten einzelner Services einen komplexen Prozess zu realisieren.

Auf www.webservices.org[12] ist folgende kurze Definition zu lesen:
”Webservices are encapsulated, loosely coupled contracted functions offered via standard protocols” [Ada01a]

und

”A Web Service is an application accessible using standard Internet protocols. They are components, and represent black-box functionality. They are NOT accessed via standard object-model-specific protocols, such as IIOP. Web Services are accessed over Web protocols such as Hypertext Transfer Protocol (HTTP) and data formats such as Extensible Markup Language (XML). A Web Service interface is defined strictly in terms of the messages the Web Service accepts and generates in response.” [Ada01b]

Hier wird ein Web Service als eine über Standard-Internetprotokolle erreichbare Anwendung beschrieben. Es sind einzelne Komponenten, die als eine Art Black-Box bestimmte Funktionen (einen bestimmten Dienst) zur

[10] Die Übersetzung ”entkoppelt” bedeutet in diesem Kontext, dass der eine Web Service nichts von den Implementationsdetails des anderem wissen muss, um mit ihm zu interagieren. Es muss lediglich die Schnittstelle bekannt sein. So ist es möglich, dynamisch mit ihnen zu interagieren.

[11] Unter Cross-Technologie versteht man die Programmentwicklung auf einer Plattform, die nicht der Zielplattform entspricht. Ein bekannter Begriff ist der Cross-Compiler, der Maschinencode für Plattformen erzeugen kann, die nicht der Plattform entsprechen, auf der das Kompilieren stattfindet.

[12] Unabhängige Web Service Community

Verfügung stellen. Sie sind nur über Web Protokolle (wie z.B. HTTP) und nicht über Objekt-Modell spezifische Protokolle erreichbar. Als Datenformat wird XML zugrunde gelegt.

Sun Microsystems hat folgende Definition:

”Web services are software components that can be spontaneously discovered, combined, and recombined to provide a solution to the user’s problem/request. The Java[tm] language and XML are the prominent technologies for Web services.” [Sun01c]

Web Services sind Softwarekomponenten die dynamisch gefunden und kombiniert werden können, um das Problem des Dienstnehmers zu lösen. Die Programmiersprache Java und XML sind bevorzugte Technologien, um Web Services zu realisieren.

Microsoft definiert Web Services folgendermaßen:

”A Web Service is a unit of application logic providing data and services to other applications. Applications access Web Services via ubiquitous[13] Web protocols and data formats such as HTTP, XML, and SOAP, with no need to worry about how each Web Service is implemented. Web Services combine the best aspects of component-based development and the Web, and are a cornerstone of the Microsoft .NET programming model.” [Mic01b]

Microsoft sieht Web Services als eine Einheit die Geschäftslogik kapselt und ihre Daten und Funktionen anderen Anwendungen zur Verfügung stellt. Die Anwendung wird über Web Protokolle und Web Datenformate wie HTTP, XML und SOAP angesprochen. Web Services vereinen das Beste aus komponentenbasierter Entwicklung und dem Internet. Sie sind ein Eckpfeiler der .NET Philosophie von Microsoft.

Es gibt also eine ganze Reihe unterschiedlicher Definitionen, die aber alle in den Kernpunkten übereinstimmen. Aus diesen Gemeinsamkeiten ergibt sich für den Autor folgende Definition:

Web Services sind kontextsensitive (vgl. Kapitel 4.5.4) Dienste, die über weit verbreitete Standard Internet Protokolle und Datenformate bereitgestellt werden. Sie beschreiben sich selbst und können daher spontan mit anderen Web Services in Aktion treten. Durch die Kombination verschiedener Services ist es möglich, komplexe Prozesse zu realisieren. Web Services sind plattform- und programmiersprachenunabhängig, da nach außen hin nur die Schnittstelle zu sehen ist und die Details der Implementierung verborgen sind. Aufgrund dieser Interoperabilität sind sie als Integrationstechnologie anzusehen. Web Services sind kein Produkt, sondern eine dienstorientierte Architektur.

Web Services sind primär für Anwendungsintegration und nicht als Endbenutzerdienste gedacht. Das schließt aber nicht aus, dass für den Endbenutzer interessante und sinnvolle Dienste über eine entsprechende Mensch-Maschine Schnittstelle (z.B. Browser) zur Verfügung gestellt werden (vgl. Kapitel 4.1).

3.2 Moderne Konzepte verteilter Systeme

In diesem Kapitel werden zunächst die Eigenschaften von verteilten Systemen besprochen. Danach werden einige aktuell vorherrschende Technologien und deren Zusammenhänge erläutert. Schließlich soll versucht werden, die Web Service Technologie innerhalb der betrachteten Technologien einzuordnen.

[13] Anm. d. Autors: ”Unter den beiden oft äquivalent gebrauchten Begriffen ”Pervasive Computing” und ”Ubiquitous Computing” wird die Allgegenwärtigkeit von Informationsverarbeitung und damit einhergehend der jederzeitige Zugriff auf Daten von beliebiger Stelle aus verstanden.” [Mat01]

3.2.1 Eigenschaften verteilter Systeme

Am Anfang der Datenverarbeitung standen die zentralisierten Systeme, die sogenannten Großrechner oder Mainframe Anlagen. Alle Computerarbeitsplätze (Terminals) waren sternförmig mit einem Großrechner verbunden. Diese Architektur hat den Vorteil, dass sie geordnet, sicher und überschaubar ist. Alle Daten liegen zentral vor und sind so leicht zu sichern. Außerdem war die Umgebung homogen, weshalb es keine Probleme mit unterschiedlichen Datenformaten und inkompatiblen Protokollen gab. Durch die hohe Leistung des Großrechners entsteht bei den Terminals der Eindruck, alle könnten gleichzeitig arbeiten. In Wirklichkeit werden die Anfragen aber sequentiell abgearbeitet. Der große Nachteil dieser Architektur ist die Verfügbarkeit, da der Mainframe Rechner einen ”single point of failure”[14] darstellt und bei seinem Ausfall das gesamte System zum Erliegen kommt.

Es entstanden die dezentralisierten Systeme. Dies bedeutet zunächst nur, dass die einzelnen Rechner jeweils untereinander verbunden sind. Es war also keine sternförmige Vernetzung mehr, sondern erinnerte mehr an eine Teil- oder Vollvermaschung. Diese Architektur war schon sehr viel robuster gegenüber Ausfällen einzelner Systeme. Jetzt gab es aber Probleme mit der Interoperabilität der Systeme untereinander. Jeder Rechner hatte ein anderes Betriebssystem, andere Datenformate oder eine unterschiedliche Namensgebung für Dateien. Aber es war möglich, nun auf gemeinsame Ressourcen und Daten zuzugreifen. Es laufen allerdings auf den einzelnen Rechnern nach wie vor eigenständige Programme ab, was bedeutet, dass Zugriffe auf gemeinsame Ressourcen nicht transparent sind. Zum Beispiel muss Nutzer A wissen auf welchem Rechner sich die Datei X befindet, um darauf zugreifen zu können.

Aus dem Wunsch heraus, Systeme transparenter zu machen, entstanden die verteilten Systeme. Hier kommunizieren nicht mehr so sehr die einzelnen Rechner miteinander, sondern die Kommunikation findet auf der Anwendungsschicht statt. Das heißt, es laufen jetzt keine eigenständigen Programme mehr auf den einzelnen Rechnern, sondern eine Anwendung auf dem Rechner kann Teil einer Gesamtanwendung sein. Typischweise wird hier eine dreischichtige Architektur verwendet (3-Tier). Das heißt, dass Visualisierung (Frontend oder View), Geschäftslogik (Middleware oder Middletier) und Datenhaltung (Backend) streng voneinander getrennt sind.

Daher ergibt sich für verteilte Systeme folgende Definition: ”Ein ’Verteiltes System’ ist eine Sammlung autonomer Rechner, welche durch ein Netz miteinander verbunden sind und über eine Software für verteilte Systeme verfügen, die integrierte Berechnungen ermöglicht.” [Spi01]

Was sind nun die Eigenschaften von verteilten Systemen. Es gibt insgesamt sechs, die als kennzeichnend gelten. Nach [Spi01] sind dies:

- **Ressourcen-Teilung (resource sharing)**
  Jeder Rechner des verteilten Systems hat Zugriff auf die Ressourcen des Gesamtsystems.
- **Offenheit (openness)**
  Es ist möglich, das verteilte System zu erweitern, da alle Schnittstellen und Protokolle offengelegt sind.
- **Gleichzeitigkeit (concurrency)**
  Mehrere Prozesse können zugleich ausgeführt werden, ohne ihren richtigen Ablauf gegenseitig zu beeinflussen.
- **Skalierbarkeit (scalability)**
  Das System kann durch zusätzliche Rechner erweitert werden und hat somit die Fähigkeit, mit den Anforderungen zu wachsen.
- **Fehlertoleranz (fault tolerance)**
  Das verteilte System bleibt funktionsfähig, auch wenn einzelne Rechner Fehlfunktionen aufweisen oder ausfallen.

[14] Kritisches, nichtredundantes System

- **Transparenz (transparency)**
  Der Nutzer nimmt das System als Einheit und nicht als System aus zusammengesetzten Komponenten wahr.

3.2.2 Zusammenhang der einzelnen Technologien

Im folgenden wird auf einige aktuelle Technologien und ihren Zusammenhang eingegangen, mit denen es möglich ist, verteilte Systeme zu realisieren.

- **Sockets**
  Sockets sind die Grundlage für eine Kommunikation über Rechnergrenzen hinweg. Sie wurden 1981 in Berkeley entwickelt. Ein Socket wird beschrieben aus der Kombination von IP-Adresse, Port und dem Protokoll. Sie stellen für sich allein genommen kein verteiltes System dar, da die Kommunikation nicht auf der Anwendungsschicht geschieht, sondern auf der Transportschicht[15]. Sie werden hier aufgrund ihrer Wichtigkeit für verteilte Systeme kurz genannt.

- **DCE**[16]
  Einer der ersten Ansätze, um verteilte Systeme zu realisieren war DCE (Distributed Computing Environment). DCE entstand Ende der 80er Jahre und hatte zum Ziel, auf allen Plattformen eine einheitliche API zur Verfügung zu stellen. Es wurde von der Open Software Foundation (OSF) ins Leben gerufen. DCE ist zwischen dem Betriebssystem und der Anwendungsschicht einzuordnen. Daher kann DCE als Middleware-Technologie bezeichnet werden. Zweck der DCE ist es, transparente verteilte Systeme in heterogenen Netzen zu ermöglichen. DCE besteht im wesentlichen aus folgenden Komponenten:

  Wichtigeste Komponente bei DCE ist der Remote Procedure Call (vgl. Kapitel 3.2.2 RPC/RMI). Durch diesen Mechanismus wird die Kommunikation mit anderen Anwendungen ermöglicht. Die DCE Threads stellen sicher, dass mehrere RPC Aufrufe gleichzeitig stattfinden können, ohne dass eine einzelner RPC Aufruf den gesamten Rechner blockiert. Der DCE Security Service implementiert eine Benutzerverwaltung, in der die User registriert sind und in der festgelegt ist, welcher User welche Dienste nutzen darf. Mit Hilfe des DCE Directory Services kann der Nutzer einen bestimmten Dienst finden, ohne den Namen des Rechners zu kennen, der diesen Dienst anbietet. Neben diesen Komponenten existiert noch das DCE Distributed File System und der DCE Time Service. Das DCE wurde Anfang der neunziger Jahre durch CORBA verdrängt und hat sich nie richtig durchsetzen können.

- **DCOM**
  Die folgende Beschreibung von DCOM orientiert sich an [GT00, S. 47 ff.]:

  DCOM (Distributed Component Object Model) ist ein Komponentenmodell der Firma Microsoft. Es entstand 1996 aus COM (Component Object Model), welches für sie Realisierung sogenannter Verbunddokumente innerhalb der Microsoftwelt entwickelt wurde. Über COM war jedoch nur eine Kommunikation von Komponenten möglich, die sich auf derselben Maschine befanden. Mit DCOM dagegen gibt es diese Einschränkung nicht. Der Zugriff auf die Komponenten ist transparent, so dass es für den Client irrelevant ist, ob sich die Komponente, auf die er zugreifen möchte, auf dem lokalen oder einem entfernet Rechner befindet. COM/DCOM Komponenten können in jeder beliebiger Programmiersprache geschrieben werden, müssen aber in einem bestimmten binären Format[17] abgelegt werden

  ”COM [und DCOM] standardisiert den Zugriff auf das Dienstangebot einer Softwarekomponente und erreicht so Interoperabilität zwischen Komponenten.” [GT00, S. 51]. Da COM/DCOM keinen Mechanismus zur Versionierung von Komponenten besitzt, ist es nicht möglich nachträglich eine Schnittstelle einer Komponente zu ändern. Es muss eine neue Schnittstelle geschaffen und veröffentlicht werden. Weiterhin haben DCOM sowie alle Varianten den erheblichen Nachteil, dass sie nur auf Windows Betriebssystemen lauffähig sind. Es gibt zwar Kompatibilitätsprodukte, die DCOM und CORBA verbinden, aber laut

[15] Nach ISO/OSI Referenzmodell Schicht 4
[16] nach [Lai01] und [Mac01]
[17] vtable

[Knu01, S. 33] sind sie aus mehreren Gründen kaum einsetzbar. Somit hat DCOM weitreichende Einschränkungen was die Interoperabilität betrifft.

- **RPC/RMI**
  RPC steht für Remote Procedure Call und beschreibt eine Architektur für entfernte Funktionsaufrufe in der prozeduralen Welt. RMI ist die Abkürzung für Remote Method Invocation und beschreibt eine Architektur für entfernte Methodenaufrufe in der objektorientierten Welt.

  Diese beiden Architekturen ermöglichen dem Client in einem Client/Server Umfeld Funktionen bzw. Methoden einer auf dem Server befindlichen Komponente aufzurufen. Die Funktion bzw. Methode wird dann innerhalb des Adressraumes des Servers ausgeführt. Es werden vom Compiler neben den normalen Dateien sogenannte Stubs und Skeletons erzeugt. Stubs sind clientseitige Proxy-Objekte, die die gleichen Funktions- bzw. Methodensignaturen besitzen wie das aufzurufende Objekt. Es ist lokal auf dem Client vorhanden und dient gewissermaßen als Stellvertreter. Die Funktions- bzw. Methodenimplementierungen sehen so aus, dass dort Code steht, der eine Netzwerkverbindung zum eigentlich verlangten (Remote-) Objekt auf dem Server aufbaut. Jedoch wird dieses Serverobjekt nicht direkt angesprochen, sondern der Stub verbindet sich vielmehr mit dem dazugehörenden Skeletonobjekt. Dieses analysiert den Datenstrom und führt dann die gewünschte Aktion auf dem tatsächlichen Objekt aus. Das Ergebnis wird an den Stub zurückgeliefert. Der Stub und der Skeleton verbergen die Kommunikationsdetails des RPC/RMI Protokolls.

  Dies ist eine der ältesten und verbreitetsten Arten, verteilte Anwendungen zu realisieren. Der Nachteil (wie auch beim statischen CORBA) ist, dass bei Schnittstellenänderungen (also Funktions-/Methodensignaturen) der Quellcode neu kompiliert werden muss. Eine in der Unixwelt verbreitete RPC Implementierung ist sun-rpc von SUN Microsystems. Das bekannte Network File System[18](NFS), ebenfalls von SUN Microsystems, baut sehr stark auf dem RPC Mechanismus auf. RMI für Java[19] ist im JDK ab Version 1.1 enthalten.

- **CORBA**
  CORBA (Common Object Request Broker Architecture) beschreibt eine Client/Server Architektur, mit der es möglich ist, plattform- und sprachunabhängig Softwarekomponenten und Objekte auszutauschen. CORBA wurde von der OMG (Object Management Group) spezifiziert. Ziel von CORBA ist die Integration von Anwendungen. Integraler Bestandteil von CORBA ist der ORB (Object Request Broker). ”Ein ORB ist ein Softwarebus, auf dem sich CORBA-Objekte registrieren können, um so ihre Dienste den restlichen, mit dem Bus verbundenen Parteien zur Verfügung zu stellen.”[GT00, S. 164]. Kommunikation von Anwendungen ist auch über ORB Grenzen hinweg möglich, da die ORBs miteinander interagieren.

  Die Schnittstellen der auszutauschenden Objekte wird mittels der programmiersprachenunabhängigen Interface Definition Language (IDL) beschrieben. Ein Compiler der aus den IDL-Dateien Code für die Zielsprache generiert, erzeugt auch die Stubs[20] (clientseitig) und den Skeleton (serverseitig). Das Stub/-Skeleton Prinzip ist ähnlich dem aus Kapitel 3.2.2 RPC/RMI. Dieser Ansatz wird als statisch bezeichnet, da bei einer Schnittstellenänderung der Quellcode erneut kompiliert werden muss. CORBA bietet auch einen dynamischen Ansatz, der zur Laufzeit auf unbekannte Objekte entsprechend reagieren kann. (Dynamic Invocation Interface (DII) auf der Clientseite und das Dynamic Skeleton Interface (DSI) auf der Serverseite.)

  Normalerweise ist jedes Objekt an genau einem ORB angemeldet. Da aber weltweit natürlich nicht nur ein ORB existiert, müssen die ORBs untereinander kommunizieren, um eine echte Verteilung zu erreichen. Dies geschieht ab CORBA 2.0 über IIOP (Internet Inter-ORB Protokoll). In früheren Versionen war es nicht festgelegt, über welches Protokoll die verschiedenen ORBs kommunizieren sollten. Es wurde den Herstellern überlassen, was dazu geführt hatte, dass die ORBs verschiedener Hersteller nicht miteinander kompatibel waren.

  Nach [Pie01] ist IIOP eine konkrete Implementierung von GIOP (General Inter-ORB Protocol), die eine Kommunikation über eine TCP/IP Verbindung erlaubt. GIOP ist ein abstraktes Protokoll, welches die grundlegenden Eigenschaften der Inter-ORB Kommunikation festgelegt.

[18] RFC 3010
[19] RMI über JRMP (Java Remote Method Protocol)
[20] Auch Proxy-Objekte genannt. Ein Proxy-Objekt hat die gleiche Signatur wie das Remote-Objekt welches angefordert wird.

Die Nachteile von IIOP sind:

  - Es ist ein binäres Protokoll.
  - Es ist keine Standard Internet Protokoll.
  - Es wird von den meisten Firewallsystemen abgeblockt.

- **Enterprise JavaBeans (EJB)**
  Die Enterprise JavaBeans sind ein Komponentenmodell der Firma SUN Microsystems. Sie sind ein integraler Bestandteil der J2EE[21] Architektur, die ebenfalls von SUN stammt. EJBs sind nur innerhalb eines Containers (EJB-Laufzeitsystem) lauffähig und erlauben moderne Dreischichten-Architekturen (3-Tier-Architecture). Typischerweise übernimmt die Aufgabe des Containers ein Applikation Server[22]. "Mit EJB soll die Entwicklung von verteilten, komponentenbasierten Java-Anwendungen vereinfacht werden." [GT00, S. 210]. Vereinfacht heißt in diesem Falle, dass dem Programmierer bestimmte Aufgaben wie Verteilung, Sicherheit und Transaktionsverwaltung abgenommen werden. Diese Mechanismen werden für den Programmierer transparent vom Container zur Verfügung gestellt. Dies garantiert robuste und fehlertolerante Anwendungen. Außerdem kann sich der Programmierer auf die Implementierung der Geschäftslogik konzentrieren. Nach den Spezifikationen von SUN soll es so sein, dass jede EJB in sämtlichen Containern (auch unterschiedlicher Hersteller) fehlerfrei funktioniert. Weiterhin ist mit RMI-IIOP die Möglichkeit gegeben, EJBs mit der CORBA Welt kommunizieren zu lassen.

- **Messaging**
  Eine Möglichkeit traditioneller Interprozesskommunikation (IPC, Inter Process Communication) sind die Message Queues. Diese funktionieren aber nicht über Rechnergrenzen hinweg. Dieser Nachteil wird von Produkten wie MQSeries von IBM oder Spezifikationen wie JMS (Java Message Service) aufgehoben. So wird es möglich, Nachrichten synchron oder asynchron zwischen einzelnen Anwendungen auszutauschen. Die einzelnen Anwendungen müssen nichts voneinander wissen (loosely coupled), um zu kommunizieren. Lediglich die Zielanwendung und das Nachrichtenformat müssen bekannt sein. Nach [Haa01] gibt es im wesentlichen zwei Möglichkeiten wie ein solches Messaging abläuft. Zum einen ist ein Punkt-zu-Punkt (Point-to-Point) Messaging möglich. Das heißt es gibt genau einen Sender und einen Empfänger der Nachricht. Die Nachricht wird bei Erhalt vom Empfänger bestätigt. Um mehrere Empfänger zu erreichen, wird das Publish/Subscribe Modell verwendet. Hier können sich alle, die Nachrichten über ein bestimmtes Thema (Topic) empfangen wollen, dort anmelden (Subscribe). Schickt ein Sender jetzt eine Nachricht zu einem bestimmten Thema ab, so wird diese allen registrierten Empfängern zugestellt.

- **Jini Network Technology von SUN**
  "Jini is a network architecture for the construction of distributed systems where scale, rate of change and complexity of interactions within and between networks are extremely important and cannot be satisfactorily addressed by existing technologies. Jini technology provides a flexible infrastructure for delivering services in a network and for creating spontaneous interactions between clients that use these services regardless of their hardware or software implementations." [Sun01a]

  Nach dieser Definition ist Jini eine Netzwerkarchitektur, mit der es möglich ist, verteilte Systeme zu realisieren. Diese sind skalierbar, flexibel und erlauben ein komplexes Zusammenspiel der beteiligten Netzwerkkomponenten. Dies ist sehr wichtig und kann derzeit mit keiner anderen existierenden Technologie realisiert werden. Jini stellt eine flexible Infrastruktur zur Verfügung, die eine dynamische Interaktion zwischen den Clients, die Jini nutzen, ermöglicht. Jini ist unabhängig von der Hardware oder Software der beteiligten Clients einsetzbar.

  Jini ist vom Prinzip her als eine Plattform für die Vernetzung von unterschiedlichsten Diensten gedacht und operiert mit Hilfe von RMI über JRMP. Des weiteren ist Jini eher Low-Level-basiert, d.h. es müssen Treiber für Jini-fähige Geräte existieren.

- **Peer-to-Peer Technologie**
  Alle bisher betrachteten Architekturen sind traditionelle Client-/Serversysteme. Doch spätestens seit dem

[21] Java2 Enterprise Edition

[22] z.B. BEA Weblogic, JBoss, Orion, Websphere ... . Manchmal werden sie auch als Komponenten-Transaktions-Monitore (CTM) bezeichnet.

Erfolg von Napster, Gnutella etc., ist die Peer-to-Peer Technologie wieder präsent. Peer-to-Peer Architekturen stellen das ”verteilteste System” dar, welches denkbar ist. Alle daran beteiligten Knoten (Rechner)[23] sind gleichberechtigt, autonom und jeder kann mit jedem kommunizieren. Jeder Knoten kann simultan sowohl als Server wie auch als Client agieren. Es ist also eine völlig dezentrale Architektur, in der es kein Zentrum gibt. Damit überhaupt eine sinnvolle Kommunikation stattfinden kann, müssen die einzelnen Knoten über eine gewisse Intelligenz verfügen. Um diese Behauptung und die gesamte Peer-to-Peer Architektur etwas näher zu betrachten, wird hier etwas genauer auf Gnutella eingegangen. Folgende Beschreibung lehnt sich hauptsächlich an [OK01, S. 94 ff.] an:

Gnutella entstand im März 2000 und wurde hauptsächlich von Justin Frankel und Tom Pepper entworfen. Es wurde ursprünglich entwickelt um Rezepte auszutauschen, erlangte dann aber als MP3 Tauschbörse und Alternative zu Napster einen hohen Bekanntheitsgrad. Gnutella ist ein vollständig dezentrales System, dessen Knoten eine Besonderheit aufweisen. Es ist nirgendwo vorgeschrieben, wie der Knoten auf die an ihn gerichtete Suchanfrage reagieren soll. Das heißt es ist ihm freigestellt die Suchanfrage ”irgendwie” zu interpretieren und dann ebenfalls ein nicht genau definiertes Ergebnis zurückzuliefern. Dies mag auf den ersten Blick etwas chaotisch aussehen, birgt aber ein enormes Potential in sich. Es erweckt nämlich den Eindruck, dass das Gesamtsystem eine gewisse Intelligenz besitzt, da jeder Knoten einen eigenen Kontext zugrunde legen kann. Ein Knoten vergleicht die Suchanfrage z.B. mit den Dateinamen auf der Festplatte, ein anderer durchsucht die Dateien selbst. Diese Kontextsensitivität geht jedoch noch einen Schritt weiter.

Ein Beispiel: Nehmen wir an, ein Nutzer gibt in ein solches System die Suchanfrage $1*1$ ein. Ein Knoten liefert ihm als Antwort einen URL, der auf eine HTML-Seite verweist, die sich mit dem kleinen $1*1$ beschäftigt. Ein anderer Knoten ignoriert Sonderzeichen und findet möglicherweise die Datei 11.mp3 auf der lokalen Festplatte. Ein weiterer Knoten, der den Kontext eines Taschenrechners besitzt, liefert als Ergebnis nur die 1, da er die Rechnung 1 mal 1 durchgeführt und ausgewertet hat.

Gnutella Knoten bieten also einen (nicht genau bekannten) Dienst an.

Peer-to-Peer stellt zwar eine neue Architektur dar, was die Zuständigkeit und Rollenverteilung der beteiligten Rechner und Anwendungen betrifft, definiert aber im eigentlichen Sinne keine neue Technologie für verteilte Systeme. Die meisten Peer-to-Peer Systeme verwenden ”traditionelles” Messaging für die Kommunikation zwischen den einzelnen Nodes.

Wie hängen diese Technologien zusammen?

- Sockets sind die Grundlage für die Netzwerkkommunikation.
- CORBA[24], DCOM und EJB sind Komponentenmodelle.
- EJBs und JMS (Java Message Service) gehören zur J2EE Spezifikation.
- RPC ist die Grundlage für DCE und DCOM.
- EJBs benötigen RMI um Verteilung zu realisieren.
- EJBs können mit Hilfe von ”RMI over IIOP” mit der CORBA-Welt kommunizieren[25].
- EJB, RMI, CORBA und DCOM sind verteilte Objekt-Technologien.
- CORBA ist im Gegensatz zu RMI/RPC plattform- und programmiersprachenunabhängig.
- Jini nutzt RMI.
- Peer-to-Peer ist ein neues Kommunikationsmodell, aber keine neue Technologie für verteilte Systeme.

[23] eng.: Nodes

[24] Es existieren in der Literatur widersprüchliche Aussagen darüber, ob man CORBA als ein vollständiges Komponentenmodell bezeichnen kann. Ab der Version 3.0 ist dies aber definitiv der Fall.

[25] Nicht nur EJBs steht diese Möglichkeit offen, sondern alle Java Klassen können diesen Mechanismus nutzen.

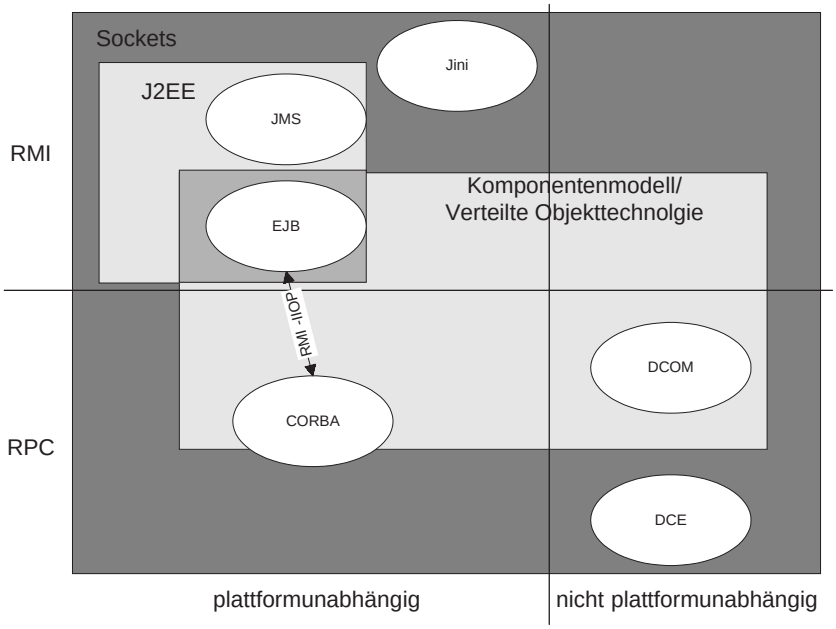

Abbildung 4: Zusammenhang der einzelnen Technologien

Nach dieser Betrachtung stellt sich die Frage, wozu Web Services innerhalb der Welt der verteilten Systeme überhaupt gebraucht werden. Es besteht kein Bedarf an einer neuen Technologie für verteilte Systeme, da die vorhandenen Technologien prinzipiell alle Problemfelder abdecken. Es besteht jedoch Bedarf in der Konsolidierung dieser Technologien. Hier kann die Web Service Technologie die Lösung bringen. Zum einen treten Web Services als Integrationstechnologie auf, die verschiedene andere Technologien verbinden kann. Zum anderen basiert diese Integration auf der vorhandenen und etablierten Infrastruktur des Internets.

3.2.3 Positionierung von Web Services

In diesem Abschnitt versucht der Verfasser, eine Einordnung der Web Service Technologie in das Umfeld der bestehenden Technologien vorzunehmen.

Web Services sind ”kein neues Komponentenmodell, sondern globale Integrationsplattform für Komponenten verschiedener Modelle.” [Pro01, S. 7]

Wie bereits im Kapitel 3.1 beschrieben, sind Web Services eine Schnittstelle zu einer Komponente, die Geschäftslogik beinhaltet. Web Services und die dazugehörenden Technologien implementieren keine Geschäftslogik, aber eine Art Broker- oder Schnittstellenlogik. Die Schnittstellenlogik sorgt z.B. dafür, eine Schnittstelle mit einer anderen kompatibel zu machen (Adapter) oder andere, komplexerer Logik zu verbergen und zu kapseln (Fasade). Diese ist in den dahinterliegenden Komponenten (z.B. EJB) abgelegt und wird über einen XML basierten RPC Mechanismus angesprochen.

Web Services sind auf den ersten Blick scheinbar ähnlich mit CORBA bzw. RPC/RMI. Dies stimmt in soweit, dass es mit beiden Technologien möglich ist, Softwarekomponenten, die auf einem anderen Rechner liegen zu manipulieren. Die entscheidenden Unterschiede sind aber:

1. Das Auffinden und die Beschreibung des Dienstes geschieht dynamisch (d.h. zur Laufzeit).

2. Die Interoperabilität fußt nicht auf einem festgelegten Protokoll, sondern wird durch Verwendung von XML als Nachrichtenformat erreicht.

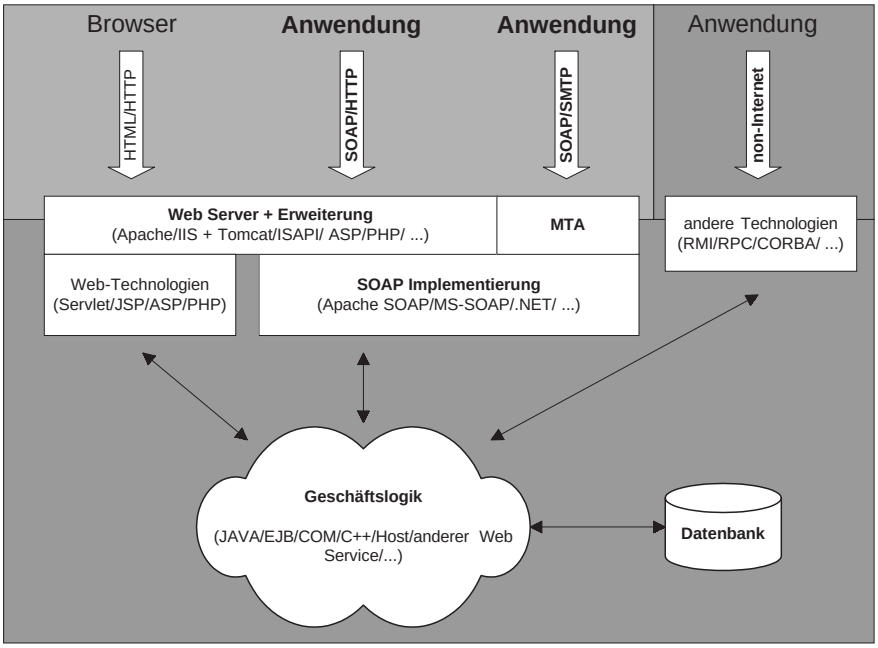

Abbildung 5: Web Services als Sicht auf das Backend

3. Web Services sind auf die Infrastruktur des Internets optimiert.

4. Die verwendeten Protokolle sind weit verbreitet und im Zusammenspiel mit Firewalls unproblematisch.

5. Web Services definieren kein Objekt-Modell.

Der Vergleich der Peer-to-Peer Architektur mit der der Web Services zeigt nur folgende Gemeinsamkeiten: Die Kontextsensitivität und die Möglichkeit aus einfachen einzelnen Web Services (bzw. Nodes) durch Komposition einen komplexen Prozess oder Workflow abzubilden.

Jini klingt dem ersten Anschein sehr nach Web Services. Die Unterschiede sind aber die Folgenden:

Web Services haben zwar den Gedanken der Serviceorientierung von Jini übernommen, aber sie stellen diese Dienste über Web orientierte Standardprotokolle wie HTTP, UDDI, SOAP und XML zur Verfügung. Jini funktioniert nur zwischen Java-Anwendungen und ist somit nicht unabhängig von der Programmiersprache. Weiterhin ist Jini eher low-level orientiert und damit weit weniger geräte- und applikationsunabhängig als Web Services. Web Services sind außerdem optimiert auf Maschine-Maschine Kommunikation, was bei Jini nicht notwendigerweise der Fall sein muss.

Web Services lassen sich aber nicht nur technologisch in eine Reihe bestehender Technologien und Protokolle einordnen, sondern sie stellen gleichzeitig auch eine Erweiterung der Infrastruktur des Internets dar. Sie machen das Internet zum Teil dienstorientiert.

Wie zu sehen ist, sind Web Services prinzipiell nichts Neues. Sowohl der Gedanke der Dienstorientierung wie auch die zugrunde liegenden Protokolle und Technologien (Internet Transportprotokolle, XML) existieren schon seit Jahren. Was aber neu ist, ist die Zusammenstellung der einzelnen Technologien und die durchgängige Verwendung von XML. Dadurch wird die zuvor nicht dagewesene Interoperabilität und Integrationsfähigkeit erreicht. Neu ist außerdem die Tatsache, dass ausschließlich die Infrastruktur und die Protokolle des Internets genutzt werden. Die Verwendung von offen und bewährten Standards verleiht den Web Services ihre Flexibilität und Stabilität.

3.3 Wichtige Spezifikationen

In diesem Abschnitt werden die wichtigsten Spezifikationen der Web Services und aller beteiligten Technologien vorgestellt:

- **Web Services allgemein**
  Die Technologie der Web Services an sich wird in verschiedenen White Papers der beteiligten Unternehmen beschrieben. Die drei wichtigsten sind:

  White Papers von IBM (vgl. [IBM01a])

  White Papers von SUN (vgl. [Sun01b])

  White Papers von Microsoft (vgl. [Mic01a])

  Diese Arbeit orientiert sich hauptsächlich an den beiden erstgenannten.

- **HTTP**
  Das Hyper Text Transfer Protokoll ist ein zustandsloses Protokoll, um nichtlinear verknüpfte kontextsensitive Inhalte (Hyper Text) über das Web zu realisieren. Es ist ein vom W3C standardisiertes Internet Protokoll. Die aktuelle Version ist HTTP 1.1. Spezifiziert wird HTTP 1.1 im RFC 2616 (vgl. [R. 01]).

- **XML**
  Die Extensible Markup Language ist sowohl ein universelles Format, um strukturierte Dokumente zu beschreiben, als auch eine Metasprache, mit der es möglich ist, andere Markup Sprachen zu definieren. XML ist eine Untermenge von SGML (Standard Generalized Markup Language). Jedes XML Dokument ist ein gültiges SGML Dokument. Die Vorteile von XML sind die Lesbarkeit (für Menschen und fast alle Maschinen) da alles im Unicode-Format vorliegt, die Einfachheit und der universelle Charakter. XML ist vom W3C spezifiziert und liegt in der Version 1.0 vor (vgl. [BPSMM01]).

- **SOAP**
  Das Simple Object Access Protokoll ist nach [GHMN01] ein leichtgewichtiges[26] auf XML aufbauendes Protokoll, um Informationen innerhalb einer verteilten Umgebung auszutauschen. Es findet ein XML-basiertes Messaging oder RPC statt. SOAP Nachrichten können an beliebige Transportprotokolle wie z.B. HTTP gebunden werden. SOAP liegt beim W3C zur Zeit als Working Draft in der Version 1.2 vor (vgl. [GHMN01].

- **XML-RPC**
  Dies war der Vorläufer von SOAP und spielt somit eine untergeordnete Rolle.

- **WSDL**
  Die Web Services Description Language ist ein XML Dokument, das die Schnittstelle eines Web Services beschreibt. Zur Zeit liegt WSDL als W3C Note[27] in der Version 1.1 vor.

- **WSFL**
  Die Web Services Flow Language ist eine auf XML basierende Sprache, die verschiedene Web Services verketten kann. Es gibt Kontrollstrukturen, die in Abhängigkeit von bestimmten Ereignissen den Ablauf festlegen. WSFL wurde von IBM initiiert und liegt zur Zeit in der Version 1.0 (vgl. [Ley01b]) vor.

- **UDDI**
  Das Akronym bedeutet ”Universal Description, Discovery and Integration”. UDDI liegt zur Zeit als Version 2.0 vor. Allerdings existieren für diese Spezifikation noch keine Implementierungen. Daher ist für den praktischen Einsatz die Version 1.0 relevant. UDDI wird von vom UDDI-Konsortium (vgl. [UDD01]) spezifiziert. UDDI ist ein verteiltes[28] und durchsuchbares Verzeichnis, an dem sich Web Services registrieren lassen. Nach einer Registrierung sind sie dann auffindbar und nutzbar.

[26] Minimale Anzahl zwingender Eigenschaften und Orthogonalität optionaler Eigenschaften
[27] Eine Note wird dem W3C als Vorschlag unterbreitet. Wird sie aufgenommen erfolgt der Status eines Working Drafts
[28] Alle UDDI Verzeichnisse gleichen sich untereinander ab.

- **ebXML**
  ist eine Sammlung von Spezifikationen, um einen Standard für den Informationsaustausch zwischen Unternehmen zu schaffen. Dieser soll auf Grundlage von XML stattfinden und auf längere Sicht solche Protokolle wie UN/EDIFACT[29] ablösen.

  Informationen über ebXML sind zu finden unter http://www.ebxml.org. ebXML wird im Rahmen dieser Arbeit nicht näher vertieft. Die Beziehung zwischen Web Services und ebXML läßt sich wie folgt beschreiben:

  ”Part of the WSDL description you get is how to ask for the service. It might say something like ’I speak ebXML’.” [Nel01]

  In der Beschreibung eines bestimmten Dienstes kann z.B. stehen: ”Ich spreche ebXML”.

Zwingend notwendig für die Realisierung von Web Services sind ein beliebiges Transportprotokoll, XML und SOAP. Allerdings ist ohne die Verwendung von WSDL und UDDI nur eine relativ statische Nutzung möglich. Damit das gesamtes Potential von Web Services genutzt werden kann, ist der Einsatz von WSDL und UDDI empfehlenswert.

3.4 Verfügbare Frameworks und Produkte

Die Web Service Technologie besteht hauptsächlich aus der Kombination der im letzten Kapitel beschriebenen Teiltechnologien, realen Implementierungen und Produkten. Diese Kombination ist derzeit nicht festgelegt. Daher haben die großen Unternehmen, die in die Entwicklung involviert sind, eigene Frameworks (also Kombinationsvorschläge) geschaffen. Es existieren zur Zeit fünf Frameworks für den Einsatz von Web Services.

1. IBM Web Services
2. Sun ONE
3. Microsoft .NET
4. Oracle9i von Oracle
5. HP e-speak von Hewlett Packard

Die zwei letztgenannten werden in dieser Arbeit nicht näher vertieft, da ihre Bedeutung zur Zeit eher gering ist. Der Markt wird von den drei erstgenannten dominiert.

3.4.1 IBM Web Services

IBM setzt auf seinen Applikationserver WebSphere und somit auf den J2EE Standard zur Implementierung der Business Logik. Als Entwicklungs- und Zielplattform ist jede Plattform geeignet, auf der Java zur Verfügung steht. Für die Entwicklung schlägt IBM das Web Service Toolkit und Visual Age für Java vor. Das Web Service Toolkit ist eine Sammlung verschiedener APIs und Kommandozeilenprogrammen. Mit Hilfe dieser Werkzeuge lassen sich, wenn auch noch auf eine sehr rudimentäre Art, Web Services erstellen. Das Toolkit wird im Rahmen dieser Arbeit zur prototypischen Implementierung eines Web Services benutzt und daher in Kapitel 5.4.2 näher beleuchtet.

[29] Electronic Data Interchange for Administration, Commerce and Transport

3.4.2 Sun ONE

Hier kommt der J2EE-fähige iPlanet Applikationserver von iPlanet für die Business Logik zum Einsatz. Als Entwicklungs- und Zielplattform ist ebenfalls jede Plattform geeignet, auf der Java zur Verfügung steht. Entwickelt werden kann mit Forte für Java. Programmiersprache ist bei SUN das hauseigene Produkt Java. Forte für Java ist eine auf NetBeans[30] aufbauende Entwicklungsumgebung (IDE) um Javalösungen zu entwickeln.

Die Sun ONE Philosophie sieht wie folgt aus (vgl. [Rei01]):

”Sun ONE is Sun’s recommended software platform for delivering Services on Demand.” [Rei01]

Sun ONE (Open Net Environment) ist ein kommerzielles Produkt für die Entwicklung von Web Services unter Nutzung der Programmiersprache Java. Der Unterschied zu ”normalen” Web Services ist die architekturimmanente Kontextsensitivität, die oft auch als ”smarte” Web Service Architektur bezeichnet wird.

Sun ONE besteht aus den folgenden Bausteinen (vgl. [CCVC01, S. 45], [Sar01, S. 41 ff.]):

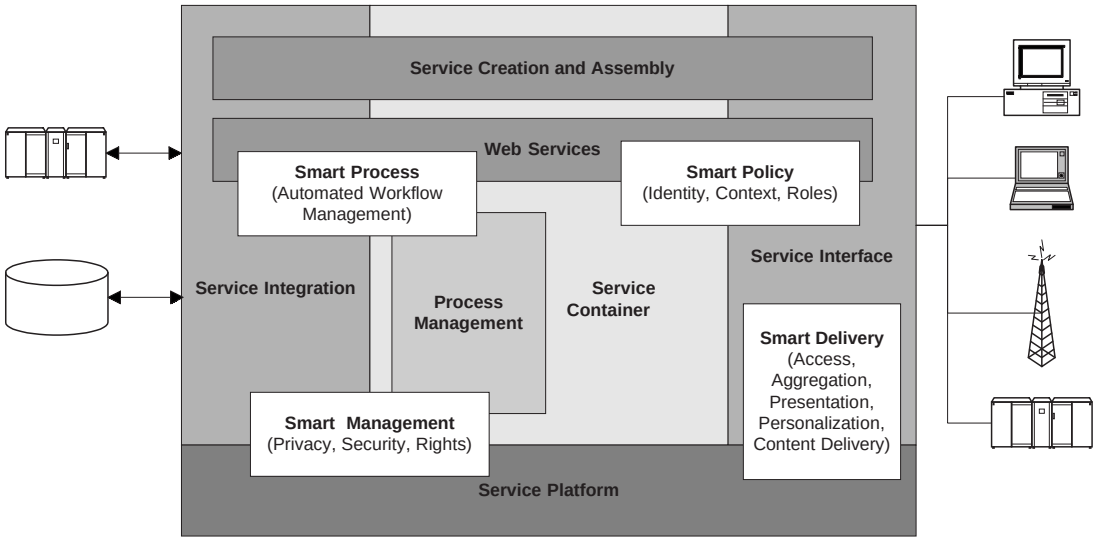

Abbildung 6: Sun ONE Architektur (vgl. [Sar01])

- **Service Creation and Assembly - Schicht**
  Entwicklungswerkzeuge und weitere Tools. Der Produktvorschlag von Sun ist die Forte Entwicklungsumgebung.

- **Service Delivery - Schicht**
  Diese Schicht übernimmt die Auslieferung des Informationen (*content*) an den Client. Dies umfasst auch die Verbindungsverwaltung, Aggregation von Inhalten, Präsentation, Personalisierung und Benachrichtigung. Der Produktvorschlag ist der iPlanet Portal Server.

- **Applications and Web Service - Schicht**
  Bereitstellung von E-Business Anwendungen als Web Services. Produktvorschlag sind die iPlanet Commerce Server.

- **Service Container - Schicht**
  Laufzeitumgebung für die Geschäftslogik und Serverdienste. Sie verwaltet u.a. Transaktionen, Persistenz undKonnektivität. Sun empfiehlt den iPlanet Application- und HTTP Server.

[30] http://www.netbeans.org

- **Service Integration - Schicht**
  Die Schicht ist zuständig für die Anbindung von Altsystemen und bestehenden Anwendungen. Produktvorschlag ist der iPlanet Integration Server und iPlanet ECXpert.

- **Identity and Policy - Schicht**
  Bereitstellung von Kontextinformationen und Sicherheit sind die Aufgaben dieser Schicht. Sun empfiehlt hier die iPlanet User Management Produkte (z.B. iPlanet Directory Server, iPlanet Certificate Management System)

- **Platform - Schicht**
  Das zugrunde liegende Betriebssystem. Solaris ist hier der Produktvorschlag von Sun. Aber auch andere UNIX sowie Microsoft Betriebssysteme sind einsetzbar.

Ein weiterer integraler Bestandteil der Sun ONE Architektur sind die JAX API's welche in Kapitel 4.4.2 besprochen werden.

Um diese Architektur so zu erweitern, dass sie "smart" (kontextsensitiv) wird, werden vier Erweiterungen hinzugefügt:

- **Smart policy**
  Koordination von Kontext, Rollen und Identität.

- **Smart delivery**
  Kontextbasierte Aggregation und Personalisierung der Web Service Resultate. Darunter fällt auch die Präsentation in verschiedenen Formaten für unterschiedliche Endgeräte.

- **Smart process**
  Koordination von Kontext und Workflow.

- **Smart management**
  Kontextabhängige Sicherheitseinstellungen. Hier kommen die Java Management Extensions[31] (JMX) zum Einsatz.

J2EE
Der J2EE Standard ist ein wichtiger Faktor, was die Web Service Entwicklung so wie sie IBM und SUN sehen, angeht. Daher soll an dieser Stelle ein kurzer Überblick gegeben werden: Die folgende Einführung in J2EE orientiert sich an [MH01].

Die J2EE Architektur von SUN stellt eine Umgebung und Technologien zur Verfügung, mit Hilfe derer es möglich ist, portable, transaktionssichere, erweiterbare und skalierbare Geschäftsanwendungen zu realisieren. Dabei wird eine n-tier Architektur zugrunde gelegt. Es existiert also eine klare Trennung von Präsentation, Geschäftslogik, die in Geschäftsobjekten gekapselt wird und der Datenhaltung. J2EE ist kein Produkt, sondern eine Sammlung verschiedener Technologien und deren Zusammenspiel. Zudem existieren Regeln wie eine J2EE konforme Architektur aufgebaut sein muss. Unter anderem spielen hier das MVC Muster[32] und sicherheitsrelevante Vorschriften eine große Rolle.

Die wichtigsten[33] involvierten Technologien sind unter anderem:

- **Enterprise JavaBeans Architektur**
  Enterprise JavaBeans sind wiederverwendbare Serverkomponenten für geschäftliche Anwendungen.

[31] Weitere Informationen sind unter http://java.sun.com/products/JavaManagement/ zu finden.
[32] Model-View-Controller Pattern
[33] Eine vollständige Liste findet sich unter [MH01, S. 408 f.].

- **JavaServer Pages**
  Mit JavaServer Pages lassen sich sehr einfach dynamische Webseiten erstellen die server- und plattformunabhängig sind.
- **Java Servlets**
  Java Servlets erweitern die serverseitigen Funktionen eines Webservers.
- **J2EE Connector**
  Mit der Connector Architektur (JCA) ist es möglich Legacy-Systeme (EIS) von Java aus anzusprechen.
- **Java Database Connectivity (JDBC)**
  Einheitlicher, abstrakter Zugriff aus Java heraus auf alle relationalen Datenbanken, für die einen JDBC-Treiber existiert.
- **Java Message Service**
  Die Java Message Service (JMS) API definiert einen Standard für den fehlertoleranten Austausch von asynchronen Nachrichten zwischen Komponenten.

Sowohl IBM als auch SUN setzen auf den J2EE Standard.

3.4.3 Microsoft .NET

Die folgende Beschreibung orientiert sich an [VJ01].

Microsoft verwendet für die Business Logik den Biztalk Server, und als Entwicklugs- und Zielplattform wird Windows 98 oder höher benötigt. Als Entwicklungsumgebung steht hier das Visual Studio .NET zur Verfügung und die unterstützten Programmiersprachen sind u.a. C#[34],Visual Basic und C++. Java wird explizit nicht unterstützt.

Microsofts .NET ist eine Sammlung von Produkten, die es ermöglicht Web Services zu entwickeln. .NET basiert auf Windows DNA (Distributed interNet Applications), die aus den Produkten Microsoft Transaction Server (MTS), COM+, Microsoft Message Queue (MSMQ) und der SQL Server Datenbank. Das .NET Framework ersetzt diese Technologien, unterstützt Web Services und zusätzliche weitere Programmiersprachen.

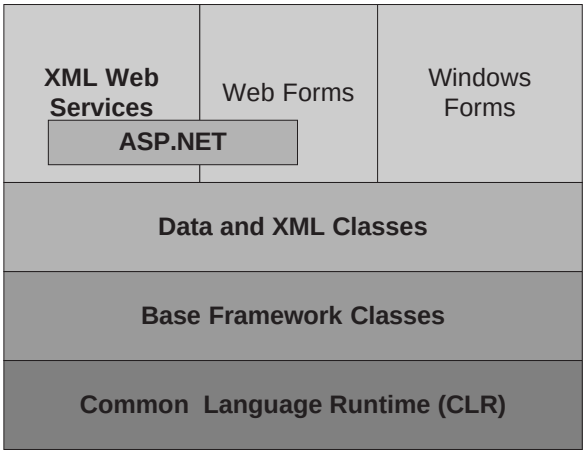

Abbildung 7: .NET Architektur (vgl. [Mic02b])

Die zulässigen Programmiersprachen, mit denen eine .NET Komponente geschrieben werden kann, sind unter anderem[35]: VB.NET, C#, C++, Phyton, Eiffel und Fortran. Diese können auch gemixt werden. Der Code wird,

[34] Neue objektorientierte Programmiersprache von Microsoft. C# (C Sharp) ist von der Syntax her ähnlich wie Java. Weitere Informationen sind unter http://msdn.microsoft.com/vstudio/nextgen/technology/csharpintro.asp zu finden.

[35] Eine vollständige Auflistung findet sich unter http://msdn.microsoft.com/vstudio/nextgen/technology/frameworkfacts.asp

analog zu Java, in eine Art Bytecode übersetzt. Diesen nennt Microsoft MSIL (Microsoft Intermediate Language). Dieser Code wird von der Common Language Runtime (CLR) interpretiert und in nativen Code übersetzt. Die CLR ist mit der Java Runtime Environment (JRE) vergleichbar.

Übersicht über die .NET Server:

- **SQL Server 2000**
  ist die relationale Datenbank von Microsoft.
- **Exchange 2000 Server**
  ist die Messaging- und Kooperationsplattform, die sowohl als Entwicklungs- wie auch als Laufzeitumgebung für Hauptgeschäftstätigkeiten dient.
- **Commerce Server 2000**
  erlaubt es, einfache E-Commerce Anwendungen zu entwickeln.
- **Application Center Server 2000**
  dient zur Verwaltung von Serverfarmen (Clustern) .
- **Host Integration Server 2000**
  erlaubt das Anbinden von Großrechnersystemen (Host).
- **Internet Security and Acceleration (ISA) Server 2000**
  stellt Firewall und Caching Funktionalität bereit.
- **BizTalk Server 2000**
  ist der XML-basierte Application Server von Microsoft.

Als Entwicklungsumgebung kommt eine modifizierte Version von Visual Studio zum Einsatz.

3.4.4 J2EE vs. Microsoft .NET

In diesem Kapitel werden der J2EE Standard und das .NET Paradigma verglichen (vgl. [VJ01]).

Hauptmerkmale

Merkmal	J2EE	.NET
Typ	Standard	Produkt
Hersteller	über 30	Microsoft
Interpreter	JRE	CLR
Dynamische Web Seiten	JSP	ASP.NET
Geschäftslogik	EJB	.NET Managed Components
Datenbankzugriff	JDBC	ADO.NET
SOAP, WSDL, UDDI Unterstützung	ja	ja

Tabelle 1: Vergleich der Hauptmerkmale von J2EE und .NET

Entwicklungszeit
Will man sich am heutigen schnelllebigen Markt behaupten, spielt es eine entscheidende Rolle, wie schnell eine Lösung umgesetzt werden kann. Eine schnelle Umsetzung wird unter anderem durch RAD[36] (Rapid Application Development) und durch Wiederverwendung von Code erreicht. Auch eine Single-Vendor Solution (die Nutzung aller erforderlichen Komponenten von einem Hersteller) trägt zu einer kürzeren Entwicklungszeit bei, da die Produkte und Lösungen untereinander kompatibel oder integrierbar sind.

J2EE bietet einige Features in dieser Hinsicht an, die in .NET nicht zu finden sind. Diese sind abstrakte Persistenzmechanismen (Entity-Beans) mit denen es möglich ist, auf persistente Daten zuzugreifen, ohne den Zugriff auf die Datenbank programmieren zu müssen.

.NET auf der anderen Seite bietet folgende Merkmale, die nicht in J2EE zu finden sind. ASP (Active Server Pages), die Microsoft Analogie zu JSP, kann endgeräteunabhänige Seiten generieren, ohne dass neuer Code geschrieben werden muss. Microsoft bietet außerdem die Queued Components an, die der J2EE Analogie MessageDriven Beans[37] überlegen sind.

Unterstützung existierender Systeme
J2EE stellt hierzu die Connector Architektur sowie JMS, Web Services, eine CORBA Schnittstelle und JNI (Java Native Interface) zur Verfügung. .NET integriert Legacy-Systeme mit Hilfe des Host Integration Server. Weiterhin können auch der COM Transaction Integrator, MSMQ und der Biztalk Server genutzt werden, um beispielsweise Systeme anzubinden, die auf MQSeries (von IBM) oder EDI aufbauen. Die Middleware-Company bewertet die Integrationsfähigkeiten von J2EE besser als die von .NET.

Stabilität
Die .NET Produkte, besonders die CLR und C#, sind noch sehr junge Technologien. Java und die J2EE Spezifikation existieren dagegen schon länger und sind akzeptierte Standards. .NET nimmt tiefe Veränderungen in der gesamten Systemarchitektur von Windows vor. Als Beispiel sei hier genannt, dass das DLL Prinzip aufgegeben wird. Das ist zwar vielversprechend, aber zur Zeit noch sehr riskant.

Programmiersprachen
J2EE unterstützt lediglich die Programmiersprache Java. Es können über CORBA andere Komponenten angebunden werden, die in anderen Programmiersprachen geschrieben wurden. Aber diese Sprachen können nicht direkt mit Java gemixt werden.

.NET unterstützt fast alle wichtigen Programmiersprachen außer Java. Vor allen Dingen die neu entwickelte objektorientierte Programmiersprache C# rückt hier in den Vordergrund. Sie ist syntaktisch relativ ähnlich mit Java, aber nicht plattformunabhängig, da die CLR nur für Windows existiert. Alle diese unterstützten Programmiersprachen können (auch innerhalb einer einzelnen Komponente) vermischt werden.

Dies hat auf der einen Seite den Vorteil, für jedes atomare Problem die richtige Sprache wählen zu können, mit der man dieses Problem am besten lösen kann. Auf der anderen Seite aber, macht es den Code unleserlich, die Projekte unüberschaubar und führt letztendlich zu ”Spaghetti Code” (vor allem da prozedurale und objektorientierte Sprachen vermischt werden können). Weiterhin wird ein wesentlich höheres Know-how und jeweils spezialisierte Entwickler für jede dieser verwendeten Sprachen benötigt. Dies kann große Probleme bei der Teamarbeit und beim Wissensaustausch der Entwickler verursachen.

Migration
Innerhalb der J2EE Plattform ist eine Migration nicht besonders problematisch, da hier lediglich für den Bereich, der Web Services betrifft, neuer Code geschrieben werden muss. Gleiches gilt für eine Anbindung von Fremdsystemen mit Hilfe der Java Connector Architektur (JCA).

[36] Unterstützt wird dies durch CASE-Tools und integrierte Entwicklungsumgebungen.
[37] seit EJB 2.0

Das sieht bei Microsofts .NET Framework anders aus, da hier der schon angesprochene Paradigmenwechsel, der mit Einführung von .NET vollzogen wird, ebenfalls überwunden werden muss. Mark Driver von Gartner bezeichnet .NET als "...fundamentally new platform" und vergleicht eine Migration bestehender Microsoft Lösungen hin zu .NET als "... more drastic than the switch from MS-DOS to Windows." [VJ01]

Es gibt zur Zeit noch keine Werkzeuge, die es ermöglichen, älteren VB Code und Systeme, die auf COM+ und MTS basieren, .NET-fähig zu machen. Mark Driver ist der Meinung, dass "...developers will have to rewrite as much as 60 percent of the code for some existing Windows applications if they want them to take advantage of Microsoft's .NET platform ..." [VJ01]

Der Code muss vor allen Dingen deshalb neu geschrieben werden, damit er von der CLR interpretierbar ist.

Portabilität
Da J2EE vollständig auf Java und dessen Grundlage der Plattformunabhängigkeit basiert, gibt es keine Einschränkungen im Punkt Portabilität. Überall dort wo eine JVM existiert, ist J2EE einsatzfähig. Das schließt alle Windows Plattformen und Mainframe Systeme ein. Portabilität wird aber nicht nur über Plattformen hinweg garantiert, sondern auf der Grundlage das J2EE auch über Herstellergrenzen hinweg, ein offener und akzeptierter Standard ist.

.NET läuft ausschließlich auf Windows Plattformen und ist somit in keiner Hinsicht portabel.

Die Grundlage einer Entscheidung sollten aber die Bedürfnisse, das Umfeld und die Historie des Unternehmens sein. Sind alle Systeme schon immer Windows basiert und ist auch das Umfeld (z.B. Zulieferer) relativ homogen, so hat .NET durchaus seine Berechtigung und Vorteile.

Web Service Support
Die gängigen Protokolle und Technologien wie SOAP, UDDI und WSDL werden von J2EE und .NET gleichermaßen unterstützt. Der Unterschied liegt darin, dass .NET bisher ebXML nicht unterstützt. Da dies aber nach Ansicht der Middleware-Company ein in Zukunft sehr wichtiger Standard für B2B Kommunikation werden kann, muss es als erheblicher Nachteil des .NET Frameworks gewertet werden.

Kosten
Innerhalb des J2EE Umfelds finden sich zur Zeit über dreißig Applikationserver in jeder Preisklasse. Hier hat man die Freiheit sich zu entscheiden und die Produkte bzw. deren Kosten an die Erfordernisse anzupassen. Dies gilt ebenso für die Hardware.

Bei .NET ist Hard- und Software von vornherein festgelegt. Für kleine Projekte kann dies überdimensioniert bzw. viel zu teuer sein. Für große Projekte reicht möglicherweise irgendwann die Leistung der Hardware nicht mehr aus.

3.5 Allgemeiner Entwicklungsstand

Da das gesamte Konzept der Web Services noch relativ jung ist, befinden sich die meisten Spezifikationen und Produkte noch in der Entwicklungsphase. Die Spezifikationen und Whitepapers sind den Implementierungen noch weit voraus. Als Beispiel sei hier UDDI genannt. Die Spezifikation befindet sich bereits in der Version 2.0 während zur Zeit nur Implementationen für die Version 1.0 existieren. Es ist jedoch so, dass eine sehr hohe Entwicklungsgeschwindigkeit erkennbar ist. Ob sich die Web Services auf dem Markt durchsetzen werden, bleibt abzuwarten. Aus technischer Sicht ist noch einiges zu tun, aber es scheint ein breites Interesse zu geben. Wichtig ist, dass die Standardisierung von WSDL und UDDI erfolgt. Diese zwei Teiltechnologien sind noch die schwachen Glieder in der Kette. Höherwertige Funktionalitäten, wie z.B. die Verkettung von Diensten (WSFL), Sicherheit, Transaktionen, usw. sind nur in Ansätzen angedacht und in ersten Papieren oder Vorschlägen niedergeschrieben. Es finden sich kaum Praxis-/Projektberichte, in denen die Web Service Technologie bisher im

größeren Umfang eingesetzt wurde. Die Entwicklungswerkzeuge für die Programmierung von Web Services sind dagegen schon recht gut, wenn auch die einzelnen Toolkits noch Probleme mit neueren Protokollspezifikationen (z.B. SOAP 1.2) haben. Außerdem sind die Toolkits zum Teil noch rudimentär und stützen sich auf Kommandozeilenprogramme (z.B. IBM Web Service Toolkit Version 2.4).

3.6 Verfügbare Services

Die bisher im Internet verfügbaren Web Services sind meist noch sehr einfach und oft noch im Test Stadium. Unter folgenden Adressen sind Web Services zu finden:

- **XMethods**
  XMethods führt eine Liste mit derzeit verfügbaren Web Services. Des weiteren werden SOAP Interoperabilitätstests sowie Diskussionsforen angeboten. Die Webseite wird seit August 2000 von Tony Hong and James Hong betrieben.
  http://www.xmethods.com/
  UDDI Adresse: http://services.xmethods.net:80/glue/inquire/uddi

- **IBM (www.ibm.com)**
  In der UDDI Registry von IBM sind produktive Web Services zu finden. Unternehmen die Dienste anbieten wollen, können dort (zur Zeit noch kostenlos) ihre Dienste publizieren.
  UDDI Adresse: http://www-3.ibm.com/services/uddi/inquiryapi

- **Microsoft (www.microsoft.com)**
  Hier gilt das gleiche wie für die IBM Registry. Microsoft bietet darüber hinaus noch seine ”Hailstrom” und ”Passport” Dienste an. Dies sind Standard Dienste wie z.B. Kalenderfunktionen, E-Mail, Lesezeichen und Authentifizierung.
  UDDI Adresse: http://uddi.microsoft.com/inquire
  .NET My Services (Hailstrom): http://www.microsoft.com/NET/netmyservices_snapshot.asp
  .NET Passport: http://www.passport.com

- **Shinka AG**
  Einfacher Währungsrechner der über den Browser bedient werden kann. Interessant ist hier, dass die gesamte SOAP-Kommunikation mitverfolgt werden kann.
  Web Service Demo: http://www.shinkatech.com:8080/shinka_central/jsp/index.jsp

Um in einem UDDI Verzeichnis nach Web Services suchen zu können, wird eine UDDI Implementierung benötigt. Unter http://www.soapclient.com/uddisearch.html ist ein UDDI Web-Interface zu finden, welches sofort einsetzbar ist. Es muss keine Software heruntergeladen oder installiert werden.

3.7 Probleme

Es ist bisher noch nicht abzusehen, welche Standards oder welche Produkte von welchem Hersteller sich hier durchsetzen werden. Bisher hat jedes Produkt oder jeder Standard noch Defizite, was die Unterstützung von Web Services angeht. Auf dem umkämpften Markt sind IBM, Microsoft, BEA und Sun Faktoren, die die Zukunft entscheidend beeinflussen werden.

Weiterhin stellt sich das Problem, dass die den Web Services zugrundeliegenden Technologien und Protokolle zum Teil noch nicht standardisiert sind oder keine vollständige Implementationen vorliegen. Für die Themen Sicherheit, Quality of Service, Transaktionen und Kontextsensitivität gibt es zwar Lösungsansätze und Technologievorschläge, aber keine etablierten Standards oder de-facto Standards (vgl. Kapitel 4.5). Die verfügbaren Toolkits zur Entwicklung von Web Services sind zum Teil umständlich zu bedienen (Kommandozeilenprogramme) und zeigen noch erhebliche Kompatibilitätsprobleme mit anderen Toolkits auf (vgl. [See01]). Es fehlt für

die kommerziellen Nutzung ein Modell zur Authentifizierung und Abrechnung, der in Anspruch genommenen Dienste. Mit Microsofts ”Passport” existiert hier ein Lösungsansatz. Dieser hat allerdings zur Zeit noch erhebliche Sicherheitsprobleme (vgl. [Han01]). Unternehmen müssten ihre Dienste zur Zeit noch selbst anbieten, da es bisher noch keine spezialisierten ”Web Service Provider” gibt.

Die Beschreibung von Web Services geschieht ausschließlich syntaktisch. Für eine automatisierte und volldynamische Interaktion zwischen Anwendungen reicht dies jedoch nicht aus. Es sind zusätzlich semantische Informationen notwendig. Das W3C macht hier mit dem ”Semantic Web”[38] Ansatz einen Schritt in die richtige Richtung.

Zusammenfassend läßt sich sagen, dass die ganze Technologie noch relativ jung und unausgereift ist. Es existieren kaum praktische Erfahrungsberichte, was sich aber in den nächsten Monaten ändern dürfte.

[38] http://www.w3.org/2001/sw/

4 Architektur von Web Services

In diesem Kapitel wird die Architektur von Web Services detailliert besprochen. Dazu wird zunächst das heutige Web-Paradigma mit dem des dienstorientierten Paradigmas verglichen. Danach wird auf die Eigenschaften und auf die verschiedenen Rollen innerhalb der Web Service Architektur eingegangen. Der Kern der Web Service Architektur besteht aus dem Protokollstack, der den Zusammenhang der einzelnen Technologien und Protokolle, aus denen die Web Services bestehen, definiert. Es folgt eine Betrachtung weiterer Eigenschaften, die für komplexe Geschäftsprozesse Voraussetzung sind. Dazu zählen insbesondere die Sicherheit und Transaktionsintegrität. Um komplexe Funktionen realisieren zu können, ist es notwendig, einzelne weniger komplexe Web Services verketten zu können. Wie so eine Verkettung aussehen kann und was dafür notwendig ist, wird am Ende des Kapitels besprochen.

4.1 Vergleich zum heutigen Web-Paradigma

”What the Web did for program-to-user interactions, Web Services are poised to do for program[-]to- program interactions.” [Kre01, S. 6]

Web Services sind auf dem Weg, das für die Interapplikationskommunikation zu werden, was das Web für die Mensch-Maschine Kommunikation war.

Das Internet in seiner heutigen Struktur ist dokumentenorientiert und daher als ein datengetriebenes Modell zu bezeichnen. Nach [Sta01, S. 13] besteht ein architektonischer Bruch zwischen dem Frontend (hier: Browser und Web Server) und dem Backend (hier: hinter dem Web Server liegende Funktionalität). Die Geschäftslogik findet ausschließlich im Backend statt, während das Frontend lediglich für den Datenaustausch zuständig ist. Die Verbindung beider Bereiche findet über herstellerabhängige Standards (z.B. JSP und ASP[39]) statt. Für diese Integration von Middleware und Web verbringen die Entwickler viel Zeit, statt sich auf die Geschäftslogik zu konzentrieren. Aber es ist dennoch kein direkter Zugriff auf Funktionalität möglich, da die Web Server keine Funktionen exportieren können. Für menschliche Benutzer stellt dieses datengetriebene Modell kein Problem dar, da Menschen die zurückgelieferten Ergebnisse problemlos interpretieren können.

Es macht einen großen Unterschied, ob Programme mit Menschen oder mit anderen Programmen kommunizieren müssen. Zwar gelten für die Mensch-Maschine Kommunikation auch Regeln, aber diese Kombination ist wesentlich fehlertoleranter. Der Mensch hat aufgrund seiner Intelligenz die Fähigkeit mit wenig strukturierten Informationen umzugehen. Er kann auf Unbekanntes flexibel reagieren. Wenn er etwas nicht versteht, kann er es ausprobieren. Er kann sehr schnell lernen und sogar fehlende Informationen kompensieren. All dies kann eine Maschine (bzw. eine Programm) in der Regel nicht. Maschinen brauchen klar strukturierte Daten die exakt definiert sind.

Aber eine maschinelle Verarbeitung ist nicht möglich, da im heutigen Web-Paradigma das Format und die Struktur dieser Ergebnisse nicht standardisiert sind. Möchte man z.B. bestimmte Informationen aus einer Webseite extrahieren, so muss das zurückgelieferte HTML mit einem Parser syntaktisch analysiert werden. Dies geschieht meist mit Hilfe regulärer Ausdrücke. Dieses Vorgehen ist nicht nur langsam, sondern auch sehr anfällig gegen Veränderungen. Ändert der Betreiber der Webseite den Aufbau des HTML Codes, so muss auch der Parser umgeschrieben werden.

Wie ist es nun möglich, diese fehlende Automatisierung von Web-Interaktion zu erreichen. Die Lösung besteht darin, nicht mehr nur Dokumente (also Daten) anzubieten, sondern eine Architektur zu schaffen, die Funktionen (Dienste) über eine standardisierte Schnittstelle bereitstellen kann. Dabei hat man sich die bereits bestehende Infrastruktur des Internets zunutze gemacht und im Wesentlichen nichts neues erfunden, sondern nur bereits vorhandene Technologien auf neue Art und Weise verbunden.

Web Services erreichen genau das und gehen dabei sogar noch einen Schritt weiter: Die Schnittstelle, über die in

[39] JSP: Java Server Pages (Sun)
ASP: Active Server Pages (Microsoft)

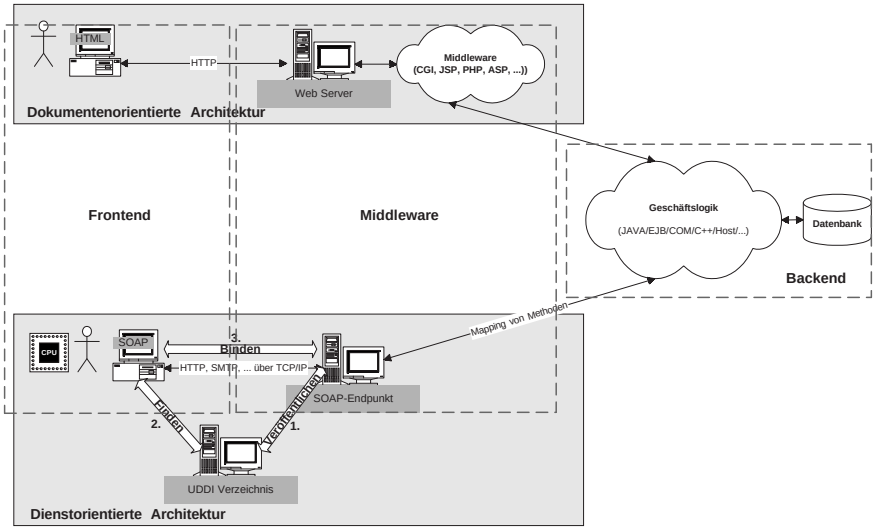

Abbildung 8: Vergleich von dokumenten- und dienstorientierter Architektur im Internet

Zukunft kommuniziert wird, basiert auf Internet Standards und beschreibt sich selbst. Dies könnte so etwas wie eine Universalschnittstelle werden. Wer diese implementiert, stellt sicher, dass nahezu jede andere Anwendung mit ihm kommunizieren kann. Da diese Schnittstelle auf XML basiert, ist sie (wie bereits erwähnt) unabhängig von der Plattform und auch unabhängig von der verwendeten Programmiersprache, in der die Anwendung implementiert ist.

Das Internet vollzieht eine Architekturerweiterung von einem Medium, auf dem Dokumente zur Verfügung gestellt, hin zu einem Medium, auf dem Dienste (Web Services) angeboten werden. Diese Web Services reichen von einfachen Diensten, wie Kreditkartenprüfung, bis zu komplexen Geschäftsprozessen, wie die Erteilung eines Kredits. Das Internet als Kommunikationsmedium für datengetriebene Informationen wird aber weiterhin wichtig und ubiquitär bleiben.

Software wird nicht mehr in Anwendungspaketen auf dem eigenen Rechner sondern in Form von Diensten über das Internet zur Verfügung gestellt. Den Services liegt eine Service Oriented Architecture (SOA) zugrunde, die eine noch nicht dagewesene Flexibilität in der Gestaltung neuer Dienste auf Grundlage vorhandener Dienste ermöglichen soll.

Wann ist eine Architektur serviceorientiert? (vgl. [Rol01]) Es müssen folgende Fragen beantwortet werden:

1. Wie können Dienste dynamisch (zur Laufzeit) gefunden werden? Die Lösung ist UDDI.
2. Wie sind Dienste beschaffen? Die Lösung ist WSDL.
3. Wie findet die Kommunikation statt. Die Lösung ist SOAP.

Weitere Merkmale einer serviceorientierten Architektur sind nach [Pü01, S. 3 ff.]:

1. Kopplung von Prozessen (statt von Systemen).
2. Eine lose Kopplung zwischen Client und Service.
3. Binding findet außerhalb des Anwendungsprogrammes statt.
4. Entkopplung zwischen Schnittstelle (Interface) und Implementierung.
5. Dynamisches Binden zur Laufzeit möglich.

6. Flexible Schnittstellen.

Michael Stahl [Sta01, S. 17] formuliert das neue Web-Paradigma so: ”Als naheliegender Lösungsansatz bietet es sich deshalb an, zusammenzubringen, was zusammengehört, und das Web zu einer Art Middleware zu erweitern.”

Der Zusammenhang ist in Abb. 8 dargestellt.

Ein Beispiel für den Paradigmenwechsel ist das Adressierungsschema im Internet:

Während ein URI (Uniform Resource Identifier) in der datengetriebenen Interpretation normalerweise eine bestimmte Datei referenziert, beschreibt er im dienstorientierten Modell einen Kommunikationsendpunkt. Dorthin können Daten geschickt werden, die dann innerhalb der zugrundeliegenden Geschäftslogik verarbeitet werden. Danach wird das Ergebnis an den Kommunikationsendpunkt zurückgegeben. Christian Weyer spricht von ”programmierbaren URIs” [Wey01b, S. 25]. Die Technologien und Protokolle, die hier nötig sind, um z.B. die Dienstschnittstelle am Kommunikationsendpunkt zu beschreiben, werden ausführlich in Kapitel 4.4 behandelt.

Dabei können beide Paradigmen problemlos koexistieren, da die ”Zielgruppe” völlig unterschiedlich ist. Maschinen können sich sehr schlecht im datengetriebenen Modell und Menschen sehr schlecht im dienstorientierten Modell zurechtfinden.

4.2 Aufbau und Eigenschaften

Wie wird nun diese dienstorientierte Architektur realisiert? Es wird zunächst der Aufbau und die Eigenschaften von Web Services besprochen.

4.2.1 Generelle Eigenschaften

Auf Grundlage der vorgestellten Definition im Kapitel 3.1 werden hier die allgemeinen Eigenschaften von Web Services zusammengefasst:

- Ein Web Service ist über das Internet erreichbar.
- Web Services besitzen eine XML Schnittstelle.
- Web Services beschreiben sich selbst.
- Web Services sind plattform- und technologieunabhängig.
- Web Services sind über ein Dienstverzeichnis auffindbar.
- Die Kommunikation findet über Standard Internetprotokolle statt.
- Web Services sind kombinierbar.
- Web Services können durch Schnittstellen definierte Prozesse anstoßen.

Die vielleicht wichtigsten Eigenschaften sind die Plattform- und Technologieunabhängigkeit. Nur sie sichern die Interoperabilität und die Integrationsfähigkeit. Daher werden diese zentralen Eigenschaften im Weiteren näher betrachtet.

4.2.2 Plattformunabhängigkeit

Im Kontext dieser Arbeit steht Plattformunabhängigkeit sowohl für die Unabhängigkeit von der Hardware als auch vom Betriebssystem. Plattformunabhängigkeit wird oft auch als Portabilität bezeichnet, wobei dieser Begriff etwas umfassender ist, da er oft noch zusätzlich die Migration von Anwendungen beschreibt.

Web Services sind augrund ihrer XML/SOAP Schnittstelle plattformunabhängig[40], da diese Schnittstelle alle Implementationsdetails verbirgt. Auf jeder Plattform für die HTTP und ein XML-Parser in einer beliebigen Programmiersprache zur Verfügung stehen, können Web Services genutzt werden.

Es findet keine Kommunikation statt, die binär codiert ist. Dieser Umstand ist sehr wichtig, da es hier (im Gegensatz z.B. zu DCOM) keine Konflikte mit dem zugrunde liegenden Betriebssystem und der verwendeten Hardware geben kann (z.B. Byteorder Problem).

Die Kompatibilität der Datentypen, wird durch die XML Schema Definition garantiert, die sämtliche Parameter für einzelne Datentypen festlegt.

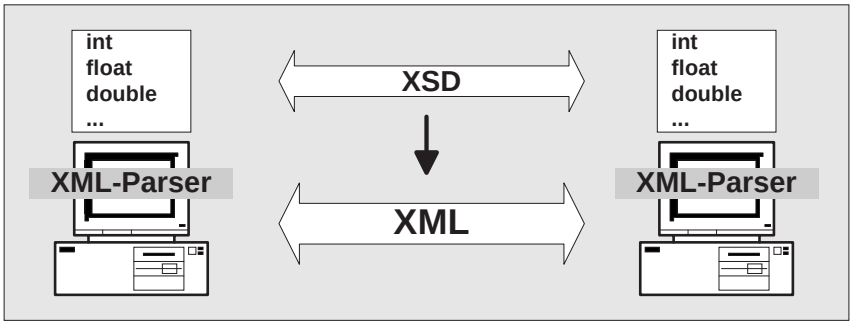

Abbildung 9: Interoperabilität durch XML

4.2.3 Technologie-/Sprachunabhängigkeit

Viel weitreichender als die reine Plattformunabhängigkeit ist die Unabhängigkeit von den verwendeten Teiltechnologien. Hierzu zählt unter anderem die Implementationssprache und das zugrundeliegende Transportprotokoll. Die Implementationssprache kann beliebig sein, solange ein XML-Parser existiert. Genauer betrachtet sind Web Services, durch die Verwendung von SOAP, vom gesamten Programmiermodell unabhängig. Ein Programmiermodell umfasst die Programmiersprache und das Paradigma (objektorientiert/prozedural).

SOAP kann prinzipiell an ein beliebiges Transportprotokoll gebunden werden. Hierzu zählen z.B. HTTP, SMTP, MQSeries, FTP, E-Mail, Fax, usw. . In der Spezifikation von SOAP 1.1 ist allerdings nur ein Binding an HTTP und SMTP beschrieben.

4.3 Teilnehmer und Funktionen

Innerhalb der Web Service Architektur gibt es drei Rollen, die an der Bereitstellung bzw. Nutzung von Services beteiligt sind. Diese werden nun im einzelnen vorgestellt (vgl. ibm-wsdc):

[40] Die Geschäftslogik, auf die der Web Service zugreift, muss hingegen nicht notwendigerweise plattformunabhängig sein. Im Falle von C# als Programmiersprache ist die Implementation an Microsoft Plattformen gebunden.

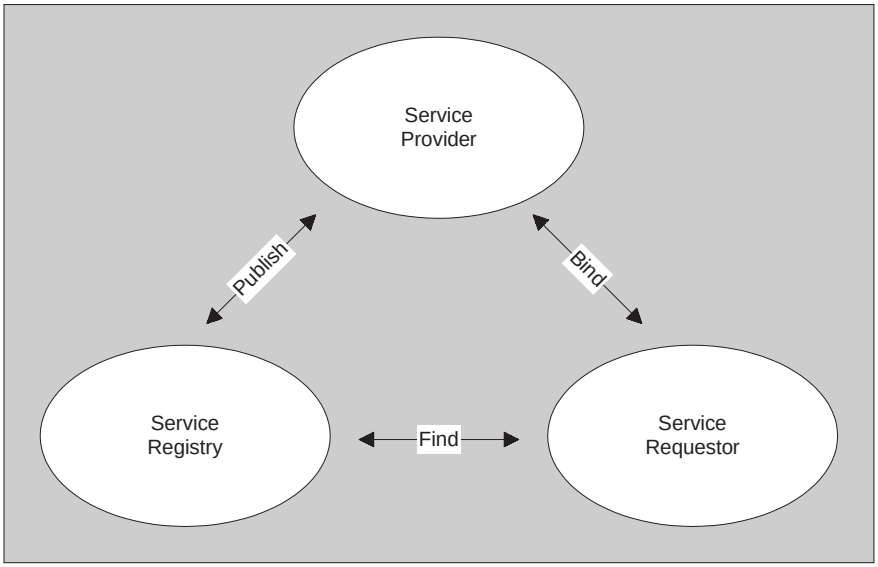

Abbildung 10: Rollen und Operationen in der Web Service Architektur (vgl. [Kre01])

4.3.1 Service Provider

Ein Service Provider bezeichnet das Unternehmen, welches einen bestimmten Service zur Verfügung stellt. Technisch gesehen ist es die Maschine, welche den Zugang zu dem Service erlaubt.

Der Service Provider muss folgende Schritte durchlaufen, um einen Service anbieten zu können:

1. **Build**
   Dieser Schritt umfasst die eigentliche Entwicklung der Geschäftlogik und/oder der Web Service Schnittstelle.

2. **Deploy**
   In dieser Phase wird die Dienstschnittstelle veröffentlicht und die Backendlogik in ihren betriebsfertigen Zustand überführt.

3. **Run**
   In diesem Zustand ist der Service betriebsbereit und er kann sowohl gefunden als auch aufgerufen werden.

4. **Manage**
   Dieser Schritt umfasst die Pflege und Wartung der Hard- und Software in Bezug auf Sicherheit, Verfügbarkeit und Performance.

Ein Dienstanbieter muss diesen Lebenszyklus an seine Voraussetzungen anpassen. Es werden folgende fünf Vorgehensweisen unterschieden:

Green Field
Geschäftslogik: besteht nicht
Web Service Interface: besteht nicht

Bei der Green Field wird das gesamte System neu entwickelt. Das heißt, dass zunächst die Applikation entwickelt und getestet werden muss, deren Funktionalität als Dienst bereitgestellt werden soll. Danach wird das Web Service Interface definiert und veröffentlicht.

Top-Down
Geschäftslogik: besteht nicht
Web Service Interface: besteht

Der Top-Down Fall trifft auf Dienstanbieter zu, die einem bereits definierten und akzeptierten Interface eine weitere Implementierung hinzufügen möchten. Dies kann sinnvoll sein, falls sich diese Implementation von bestehenden abgrenzt (z.B. schneller oder günstiger).

Bottom-Up
Geschäftslogik: besteht
Web Service Interface: besteht nicht

Dies ist sozusagen der Integrationsfall, der wahrscheinlich am häufigsten auftritt. Eine bestehende Applikation soll ein Web Service Interface erhalten, um so über Standard Protokolle via Internet zugänglich zu sein. Es werden die zu exportierenden Methoden definiert und daraus eine Dienstbeschreibung erzeugt und veröffentlicht. Diese Methode wird in Kapitel 5 für die prototypische Implementierung angewendet und dort Schritt für Schritt beschrieben.

Meet-in-the-Middle
Geschäftslogik: besteht
Web Service Interface: besteht

Der Meet-in-the-Middle Fall beschreibt folgende Situation: Es existiert eine Anwendung mit einer bestimmten Schnittstelle, die nicht mehr geändert werden kann. Es existiert außerdem noch eine (bereits veröffentlichte) Dienstbeschreibung, die ebenfalls nicht mehr geändert werden kann. Soll nun die Applikation an die inkompatible Dienstbeschreibung angebunden werden, so muss man ”sich in der Mitte treffen” und eine Brücke schaffen, die beide Schnittstellen kompatibel macht. Diese Brücke nennt man ”Wrapper”.

Compose
Ein neuer Web Service kann auch dadurch entstehen, dass bereits verfügbare Services aggregiert werden. Es muss keine eigene Geschäftslogik vorhanden sein. Ein Beispiel wäre aus einem Service, der Kreditkartennummern prüft und einem anderen der die Kreditwürdigkeit des Karteninhabers prüft, einen Service anzubieten, der sicher stellt, dass der Kunde kreditwürdig ist. Die Prüfung der Kreditkartennummer geschieht für den Nutzer des neuen (aggregierten) Web Service transparent. Diese Verkettung von Web Services wird in Kapitel 4.5.5 näher beschrieben.

4.3.2 Service Registry

Die Service Registry ist ein durchsuchbares Verzeichnis, an dem Dienste angemeldet werden können, um gefunden zu werden. Diese Rolle ist in der Web Service Architektur optional, da Dienste bzw. deren Beschreibung auch auf anderem Wege einem Nutzer zugänglich gemacht werden können. So ist es z.B. möglich die Schnittstellendefinitionen (WSDL Datei) einem interessierten Geschäftspartner per E-Mail oder FTP zukommen zu lassen. So ein Vorgehen wird als ”statisch gebunden” bezeichnet.

4.3.3 Service Requestor

Der Service Requestor ist das Unternehmen welches einen Dienst aufruft. Technisch gesehen ist es die Client-Anwendung. Auch der Service Requestor muss die Schritte *build, deploy und run* durchführen. Hier haben diese Phasen jedoch eine etwas abgewandelte Interpretation. Die zentrale Aufgabe, die der Client erfüllen muss, ist das *Binding*. Dies ist der Vorgang, in dem die Dienstbeschreibung des Service Providers benutzt wird, um ein

Proxy auf der Clientseite zu erzeugen. Über diesen Proxy können dann die Methoden aufgerufen werden, die der Service zur Verfügung stellt.

Das Binding kann auf drei unterschiedliche Arten erfolgen:

Static Binding
Ein Dienst wird dann statisch gebunden, wenn alle Spezifikationen des Dienstes bereits vor der Laufzeit bekannt sind und sich diese während der Laufzeit nicht ändern. Dies betrifft vor allen Dingen die Netzwerkadresse des Kommunikationsendpunktes und die Methodensignaturen. Das statische Binden geschieht zum Kompilationszeitpunkt. Es ist ungleich performanter[41] als ein dynamisches Binden, da die Erzeugung des Proxy Codes zur Laufzeit entfällt. Allerdings ist das System dahingehend unflexibel geworden, weil jetzt ohne manuelle Eingriffe, nicht mehr auf sich ändernde Anforderungen bezüglich der Spezifikationen des Dienstes, reagiert werden kann.

Build-time Dynamic Binding
Diese Art und Weise einen Dienst zu binden wird dann angewendet, wenn zwar der Dienst bekannt ist (d.h. Methodensignaturen, Datentypen, Transportprotokoll), der genutzt werden soll, aber der Kommunikationsendpunkt zum Kompilationszeitpunkt unbekannt ist. Es wird hier clientseitig analog zum statischen Binden zum Kompilationszeitpunkt ein Proxy erzeugt, der jedoch Code enthält, um zur Laufzeit ein Verzeichnis (z.B. UDDI) zu kontaktieren. Diese liefert die Netzwerkadresse des Endpunktes und der Dienst kann dann genutzt werden. Die Performanz ist hier niedriger als beim reinen statischen Binden, da zur Laufzeit Operationen auf einem Verzeichnisdienst ausgeführt werden müssen. Vorteil der Methode ist die Flexibilität, gleichartige Dienstimplementationen zur Laufzeit austauschen zu können.

Runtime Dynamic Binding
Das volldynamische Binden eines Dienstes unterscheidet sich vom dynamischen Binden zum Kompilationszeitpunkt dadurch, dass hier weder die Spezifikation noch die Adresse des Endpunktes vor der Laufzeit bekannt sind. Somit muss also die Dienstbeschreibung zur Laufzeit gefunden und daraus unmittelbar der Proxy erzeugt werden. Dieses volldynamische Binden ist allerdings für eine reine Maschine-Maschine Kommunikation bisher ungeeignet, da zwar die Syntax nicht aber die Semantik in der Dienstbeschreibung definiert ist. Diese Methode bietet zwar die größte Flexibilität, aber auch die geringe Performanz. Es müssen zur Laufzeit sowohl Operationen auf einem Verzeichnisdienst (das Finden der Dienstbeschreibung) ausgeführt, sowie der Proxy Code erzeugt werden.

4.4 Der Protokollstack

4.4.1 HTTP als Transportprotokoll

Das HyperText Transfer Protocol (HTTP) wurde 1991 von Tim Berners-Lee am CERN[42] entwickelt. Es war ursprünglich für eine nicht-lineare und assoziative Informationsstrukturierung (Hypertext) konzipiert worden, um den Wissenschaftlern am CERN den Zugriff und die Strukturierung großer Datenmengen zu erleichtern. Die erste HTTP Version war 0.9 . Die aktuelle Version ist 1.1 . Über HTTP wurden zunächst nur HTML (HyperText Markup Language) Dateien übertragen. HTML ist eine SGML DTD[43] und die Seitenbeschreibungssprache des WWW (World Wide Web).

HTTP ist ein Protokoll auf der Anwendungsschicht, für verteilte und gemeinschaftlich genutzte Hypermedia Informationssysteme. Es ist ein generisches und zustandsloses Protokoll und setzt typischerweise auf TCP/IP

[41] Mit Performanz ist hauptsächlich die Antwortzeit gemeint.
[42] European Organization for Nuclear Research, http://public.web.cern.ch/Public/ACHIEVEMENTS/web.html
[43] SGML steht für Standard Generalized Markup Language. DTD bedeutet Document Type Definition. Weitere Informationen unter http://www.w3.org/TR/REC-html40/sgml/dtd.html

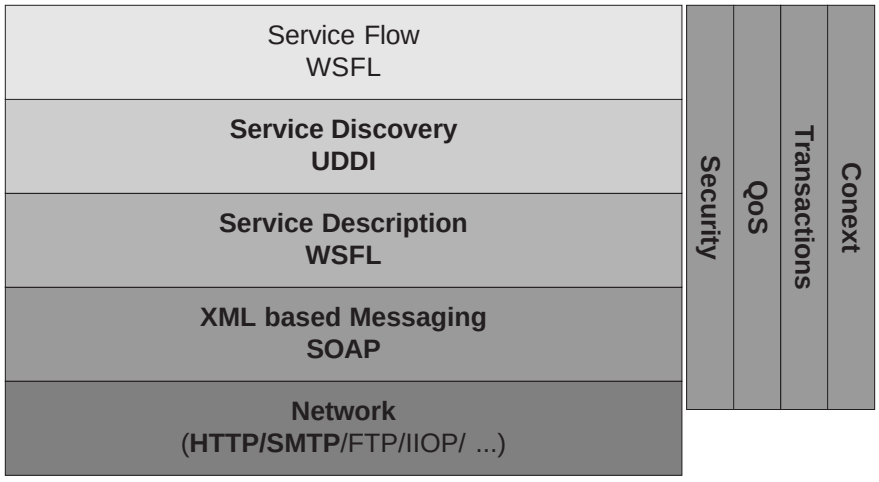

Abbildung 11: Der Web Service Protokollstack (vgl. [Kre01])

auf. Generisch bedeutet, dass es für weit mehr als nur zu Transport von HTML benutzt werden kann. Es können beliebige, auch binäre, Daten transportiert werden. Dies wird durch die Nutzung der MIME[44] erreicht. Das HTTP Protokoll arbeitet nach dem Request/Response Prinzip (Abb. 12). Weitere wichtige Eigenschaften von HTTP sind zum einen die generelle Unterstützung und Verfügbarkeit auf fast allen Systemen und zum anderen die Einfachheit des Protokolls. Dies sind die optimalen Voraussetzungen, um das de-facto Transportprotokoll für Web Services respektive SOAP zu sein.

Das Request/Response Prinzip:

1. Client baut die Verbindung auf.
2. Client sendet eine Anfrage (Request).
3. Server schickt eine Antwort (Response).
4. Server baut die Verbindung ab[45].

Da zwischen zwei solchen *Requests* oder *Responses* keine Informationen über den jeweils vorhergehenden Aufruf gespeichert werden, ist HTTP ein zustandsloses Protokoll. Abb. 12 zeigt dies, anhand einer einfachen HTTP Verbindung.

Eine HTTP Nachricht besteht immer aus einem *Header* (Kopf) und meist einem *Body* (Nutzdaten), der aber auch optional sein kann. Welche Daten nun im Header und welche im Body übertragen werden, ist davon abhängig ob es eine HTTP Anfrage oder eine HTTP Antwort ist.

```
GET /index.html HTTP/1.1
Host: de.yahoo.com
<CRLF>
```

Dies ist eine sehr einfache, gültige HTTP Anfrage. Über den Befehl **GET** wird dem Server mitgeteilt, dass der Client die dem GET-befehl nachfolgende Datei anfordert. Nach einem Leerzeichen gibt der Client an, welche HTTP Version er unterstützt. HTTP 1.1 verlangt die Angabe des Hostnamens. Mit dem abschließenden CRLF[46] wird der Header vom Body getrennt. Der Body ist in diesem Falle leer.

[44] Multipurpose Internet Mail Extensions, http://www.ietf.org/rfc/rfc2046.txt

[45] Ab HTTP 1.0 ist es möglich, persistente Verbindungen zu etablieren. Ab HTTP 1.1 ist dies sogar der Normalfall. Persistente Verbindungen dienen hauptsächlich zur Steigerung der Performanz. HTTP wird dadurch aber nicht zustandsbehaftet.

[46] Carriage Return Line Feed

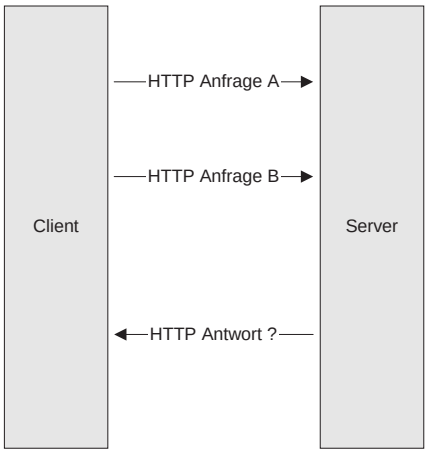

Abbildung 12: Zustandslosigkeit von HTTP

```
GET /test.jsp?name=Skywalker&vorname=Luke HTTP/1.1
Host: localhost
<CRLF>
```

Innerhalb dieser GET Anfrage werden Daten an den Server übermittelt. Da GET Anfragen immer einen leeren Body besitzen, werden Daten, die für den Server bestimmt sind, an den URI angehängt.

GET Anfragen haben zwei Nachteile: Erstens müssen die Daten speziell codiert werden, da viele Zeichen in einem URI nicht erlaubt[47] sind. Zweitens ist die Datenmenge die übertragen werden kann, oft begrenzt. Nach der HTTP Spezifikation kann die Länge eines URIs beliebig lange sein, aber viele Anwendungen limitieren den URI auf eine Länge von 255 Zeichen. Daher wird im Falle einer Datenübertragung normalerweise POST verwendet.

Hier ein Beispiel:

```
POST /test.jsp HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 31
<CRLF>
name=Skywalker&vorname=Luke
```

Die Daten werden hier im HTTP Body codiert und übertragen.

HTTP Response Die Antwort auf die erste Anfrage (s.o.) sieht folgendermaßen aus:

```
HTTP/1.1 200 OK
Date: Sun, 14 Oct 2001 14:49:49 GMT
Cache-Control: private
Expires: Sun, 14 Oct 2001 14:49:49 GMT
Connection: close
Content-Type: text/html
Set-Cookie: B=a21i0astsj9gd&b=2; <...> domain=.yahoo.com
```

[47] vgl. http://www.w3.org/Addressing/rfc1738.txt

```
<html>
<head>
<title>Yahoo! Deutschland</title>
<base href=http://de.yahoo.com/r/>
</head>
<body>

<...>

</body>
</html>
```

Die erste Zeile der Antwort enthält neben der HTTP Version, die der Server unterstützt, einen Statuscode in Form einer dreistelligen Zahl. Die wichtigsten Statuscodes sind:

- 200 : Ok (Anfrage erfolgreich)
- 3xx : Redirection (Umleitung; Die angeforderte Ressource ist vorhanden, aber unter einem anderen URI)
- 400 : Bad Request (Syntaxfehler in der Anfrage des Clients)
- 401 : Unauthorized (Client ist nicht autorisiert)
- 403 : Forbidden (Client ist nicht ermächtigt, auf die Ressource zuzugreifen)
- 404 : Not Found (Die angeforderte Ressource wurde nicht gefunden)
- 405 : Method not allowed (Die gewünschte Aktion ist nicht erlaubt)
- 500 : Internal Server Error (Interner Fehler des Servers)

Ebenfalls von Bedeutung ist die sechste Zeile, das Header Feld Content-Type, welches den MIME Typ des HTTP Bodys angibt. In diesem Fall ist es HTML. Wird z.B. eine GIF-Grafik vom Server angefordert, so ist der Content-Type image/gif. Das Bild wird im HTTP Body als Folge von Bytes übertragen (also als binärer Datenstrom). Die wichtigsten MIME-Typen sind:

- text/plain : Text (ohne Interpretation)
- text/html : Text der HTML darstellt
- text/xml : Text der XML darstellt
- image/jpeg : Ein JPEG Bild (binär)
- image/gif : Ein GIF Bild (binär)
- application/java : Eine Javaklasse im .class Format (binär)
- application/octet-stream : Unbekannter binärer Datenstrom

Bei genauer Betrachtung einer HTTP Nachricht, stellt man fest, dass HTTP einen einfachen RPC-Mechanismus repräsentiert. Dies wird besonders deutlich, wenn ein URI z.B. ein CGI-Programm referenziert, an welches nun per GET oder POST Daten geschickt werden. Diese Daten werden verarbeitet und das Ergebnis wird an den Client zurückgeschickt. Dies ist natürlich ein sehr einfacher RPC-Mechanismus, da weder komplexe Datentypen übergeben werden können noch irgendwelche Schnittstellen bekannt sind. Ferner ist der Name der Funktion oder Methode, die aufgerufen werden soll, entweder als Parameter oder direkt als URI Endpunkt definiert (vgl. [Mer01]).

Sicherheit und Authentifizierung Bisher wurde noch nichts über die Sicherheit von HTTP gesagt.

HTTP wird, wie fast alle Internetprotokolle, im Klartext übertragen und ist daher weder abhörsicher, noch vor Manipulation geschützt. Typischerweise finden auch keine Authentifizierungsvorgänge statt.

Diese Sicherheit war in den Anfangstagen des WWW gar nicht nötig, da keine schützenswerten Daten zu übertragen waren und außerdem ein nur sehr kleiner Nutzerkreis überhaupt Zugang zum WWW hatte. Doch mit der Öffnung des Netzes für kommerzielle Zwecke war es notwendig geworden, gewisse Daten wie z.B. Kreditkartennummern zu schützen. Aus diesem Umstand heraus entstand HTTPS (HTTP über SSL). Da bei HTTPS die Sicherheit durch ein darunterliegendes Protokoll realisiert wird, und daher nichts direkt mit HTTP zu tun hat, wird es im Kapitel Sicherheit (vgl. 4.5.1) näher behandelt.

HTTP überträgt Daten zwar ungeschützt, besitzt aber einen eigenen Authentifizierungsmechanismus, der in RFC 2617 beschrieben wird. Dieser schützt einen bestimmten, als solchen gekennzeichneten Bereich (realm) auf dem Server. Dies kann eine einzelne Datei sein oder ganze Verzeichnisse umfassen. Auf dem Server wird für diesen Bereich eine Kombination von Benutzername und einem Passwort hinterlegt. Möchte ein Client mittels einer HTTP Anfrage darauf zugreifen, so schickt der Server zunächst einmal ein 401 Fehlercode (Unauthorized). Daraufhin kann der Client seine Anfrage erneut stellen und ein zusätzliches Header Feld mitschicken, welches einen Benutzernamen und ein Passwort enthält. Diese Information wird auf Serverseite mit der hinterlegten Kombination aus Benutzername und Passwort verglichen und bei einer Übereinstimmung wird der Zugriff erlaubt. Stimmen diese Authentifizierungsmerkmale nicht überein, wird wiederum ein 401 Fehlercode gesendet. Ein Client muss den ersten 401 Fehlercode nicht notwendigerweise empfangen, sondern kann schon bei der ersten Anfrage den Identifizierungstoken mitschicken.

Dieses Verfahren existiert nach RFC 2617 in zwei Varianten:

- **Basic**
  Beim Basic Verfahren werden der Benutzername und das Passwort im Klartext übertragen. Dieses Verfahren ist daher gegen Angriffe wenig geschützt, wird aber von allen clientseitigen Applikationen (z.B. Browser) unterstützt. Der Authentifizierungstoken (”Authorization”) im Header der HTTP Anfrage wird folgendermaßen kodiert:

  ```
  credentials = "Basic" basic-credentials
  basic-credentials = base64-user-pass
  base64-user-pass  = <base64 encoding of user-pass,
                      except not limited to 76 char/line>
  user-pass   = userid ":" password
  userid      = *<TEXT excluding ":">
  password    = *TEXT
  ```

  Somit sieht die Authentifizierung im Header so aus:

  ```
  Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
  ```

- **Digest**
  Das Digest Verfahren funktioniert ähnlich dem Basic Verfahren, aber mit dem Unterschied das der Benutzername und das Passwort nicht im Klartext übertragen werden. Sie werden mittels eines Digest Verfahren (One-Way Hashes, z.B. MD5 oder SHA1) verschlüsselt. Der Server verschlüsselt die hinterlegten Daten ebenfalls mit dem gleichen Verfahren und vergleicht dann diese beiden Ergebnisse. Das Digest Verfahren wird an dieser Stelle nicht weiter vertieft. Der interessierte Leser sei jedoch auf das RFC 2617 verwiesen.

Das Basic und auch das Digest Verfahren sind keine wirklich sicheren Verfahren, um sensible Daten zu schützen. Dies liegt zum einen darin begründet, dass die eigentlichen Daten (HTTP Body) immer unverschlüsselt

übertragen werden und zum anderen daran, dass der Server sich gegenüber dem Client nicht identifiziert. Dies wiederum ermöglicht das sogenannte Spoofing[48].

Die gesamte Sicherheitsthematik wird im Kapitel Sicherheit (vgl. 4.5.1) detailliert behandelt. Dort wird auch besprochen, in wie weit der HTTP Authentifizierungsmechanismus im Kontext der Web Services sinnvoll ist.

4.4.2 XML als universelles Datenformat

Die Extensible Markup Language (XML) ist eine generische Sprache, um Daten zu strukturieren. XML ist ebenfalls eine generische Metasprache, um Sprachen zu definieren. Sie ist eine Untermenge[49] von SGML und jedes XML Dokument ist ebenfalls ein gültiges SGML Dokument.

”XML stellt jedoch keine echte semantische Auszeichnungssprache dar, da durch die Metasprache lediglich eine Möglichkeit zur Formulierung eigener Syntax gegeben ist. Die Bedeutung der Elemente bleibt jedoch unberücksichtigt, und kann mittels XML nicht ausgedrückt werden.” [Jec01c]

Wann ist ein Textdokument ein XML Dokument?

”An XML document is valid if it has an associated document type declaration and if the document complies with the constraints expressed in it.” [BPSMM01]

Um gültig (`valid`) sein zu können, muss ein XML Dokument zunächst wohlgeformt (`well formed`) sein. Darüber hinaus muss es eine ”Document Type Declaration” besitzen und den darin spezifizierten Regeln genügen. Dies kann eine DTD (Document Type Definition) oder ein XML Schema sein. Ein XML Dokument muss nicht notwendigerweise gültig, aber in jedem Falle wohlgeformt sein.

- **Wohlgeformt** (vgl. [BPSMM01], [Jec01c])
  Ein XML Dokument ist well formed (wohlgeformt), wenn
  - zu jedem Start-Tag[50] genau ein Ende-Tag existiert (Bei leeren Elementen können diese zu einem Tag zusammenfallen).
  - die Elementschachtelung hierarchisch korrekt ist (d.h. Elemente überlappen einander nicht).
  - es genau ein Wurzelelement gibt.
  - alle Attributwerte in einfachen oder doppelten Anführungszeichen stehen.
  - kein Start-Tag (oder Tag der ein leeres Element einleitet) zwei oder mehr Attribute desselben Namens enthält.
  - keine Kommentare oder Processing Instructions innerhalb von Tags stehen.
  - Kommentare mit einem Ausrufezeichen und genau zwei Bindestrichen beginnen und mit genau zwei Bindestrichen enden (<!– Kommentar –>).
  - die Sonderzeichen < und & nicht innerhalb von Elementinhalten oder Attributwerten auftreten.
- **Gültig**
  Ein XML Dokument ist gültig (valid, schema valid), wenn es wohlgeformt ist und den Regeln einer zugewiesenen DTD oder eines zugewiesenen XML Schemas entspricht.

[Rol01] vergleicht die Beziehung eines XML-Dokuments (Dokumenten-Instanz) zu seiner DTD oder Schema Definition mit der Beziehung von dem tatsächlichen Inhalt einer Datenbanktabelle (Tupels) zum Datenbankschema. Es findet eine Trennung von den tatsächlichen Inhalten (Instanz) und der Typinformation (DTD, XSD) von diesen Inhalten statt.

[48] Unter Spoofing versteht man in diesem Zusammenhang, dass sich ein ’feindlicher’ Server zunächst für einen anderen Server ausgibt und den Nutzer auffordert ein Passwort zu übermitteln. Da der Nutzer nicht weiss, dass er nicht mit seinem eigentlichen Ziel verbunden ist, wird er das Passwort eingeben und dieses so dem ’feindlichen’ Server zugänglich machen. Dort muss es nur noch aufgezeichnet und abgespeichert werden.

[49] auch: vereinfachte Form

[50] *Tag* ist eine generische Bezeichnung für ein beschreibendes Sprachelement. Wird oft auch *Markup* genannt.

”Each XML document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, sometimes called its ’generic identifier’ (GI), and may have a set of attribute specifications.” [BPSMM01]

XML Dokumente bestehen aus Elementen (Tags) und Attributen. Ein Attribut ist immer einem Element zugewiesen und steht in dessen Starttag. Ein Element kann null oder mehr Attribute enthalten. Ein Element kann beliebig viele Unterelemente (Kindelemente) besitzen. Die Elemente müssen hierarchisch korrekt geschachtelt sein. Ferner kann ein Element auch leer sein, oder lediglich Text enthalten.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<root>
    <ebene11>
        <ebene12 attribut1="Wert1" attribut2="Wert2">
            Hier steht Inhalt
        </ebene12>
    </ebene11>
    <ebene21>
            Hier steht ebenfalls Inhalt
    </ebene21>
    <ebene31 attribut1="Wert1" />
</root>
```

Die Struktur von XML wird durch das W3C-Dokument XML Information Set[51] definiert. Dort werden die verschiedenen strukturbildenden Bestandteile eines XML Dokumentes beschrieben. Die wichtigsten sind u.a. das ”Document Information Item”, die ”Element Information Items” und die ”Attribute Information Items”.

XML Parser Mit Hilfe von XML Parsern kann man XML Dokumente verarbeiten und so in eigene Anwendungen integrieren. Solche Parser sind die Grundlage, für jegliche Interaktion zwischen Programmen und XML Dokumenten. Diese Interaktion kann sowohl einen rein lesenden Zugriff wie auch einen schreibenden Zugriff bedeuten. Hierzu existieren zur Zeit zwei verschiedene Ansätze bzw. API Standards, die hier kurz erläutert werden sollen. Diese Beschreibung orientiert sich an [Meg01] und [W3C01a].

- **Simple API for XML (SAX)**
  SAX definiert eine einheitliche Programmierschnittstelle, um XML Dokumente zu parsen. SAX hat sich mittlerweile als de-facto Standard etabliert und entstand aus der Notwendigkeit, einheitlich (standardisiert) auf die zum Teil sehr unterschiedlichen APIs der XML Parser zugreifen zu können. SAX wurde hauptsächlich von David Megginson entwickelt. Einer der bekanntesten Parser mit einer SAX, ist der Xerces Parser von Apache (http://xml.apache.org). SAX Parser zeichnen sich durch eine hohe Verarbeitungsgeschwindigkeit und niedrigen Speicherverbrauch aus. Dies wird durch den sequentiellen Parsingprozess erreicht, da er bereits während der Verarbeitung Daten über das XML Dokument zur Verfügung stellt. Außerdem werden keine Strukturinformationen oder längere Teile des XML Dokuments im Speicher gehalten. Allerdings ist es nicht möglich, gezielt auf Teile des XML Dokuments direkt zuzugreifen, oder das Dokument zu modifizieren. Außerdem muss die Traversierung des Dokuments vom Entwickler programmiert werden, da SAX keine logische Repräsentation des Dokuments zur Verfügung stellt (vgl. [Meg01]).

  SAX Parser arbeiten ereignisorientiert. Das bedeutet, dass ein XML Dokument von Anfang bis Ende sequentiell abgearbeitet und an jeder signifikanten Stelle ein Ereignis ausgelöst wird. Diese signifikanten Stellen sind der Beginn eines Elementes, der Inhalt eines Elementes und das Ende eines Elementes. Hierzu gibt es entsprechende Callback Methoden, die vom Programmierer überschrieben werden müssen. Diesen Callbackmethoden werden alle relevanten Informationen als Parameter mitgegeben.

  Beispiel für eine solche Callbackmethode:

[51] http://www.w3.org/TR/xml-infoset/

Quellcode 4.1: SAX

```
1  public void startElement(java.lang.String uri,
2                           java.lang.String localName,
3                           java.lang.String qName,
4                           Attributes attributes)
5  {
6     System.out.println("Element "+qName+" started");
7
8     if(attribute.getLength() > 0)
9        System.out.println("Element "
10                          +qName+
11                          " fist attribute is "
12                          +attribute.getQName());
13    ...
14    ...
15 }
```

- **Document Object Model (DOM)**
  Ganz im Gegensatz zu SAX, liest ein DOM orientierter XML Parser das Dokument zunächst vollständig in den Speicher ein und stellt so eine Repräsentation der logischen Struktur zur Verfügung. Das XML Dokument wird als Baum interpretiert. Durch diesen wird navigiert und auf dessen Knoten und Blätter kann direkt zugegriffen werden. Ferner können Knoten bzw. Blätter hinzugefügt, ersetzt oder gelöscht werden. Es ist also möglich, das XML Dokument zu modifizieren. Diese Vorzüge müssen aber mit einer langsameren Verarbeitungsgeschwindigkeit und höherem Speicherverbrauch bezahlt werden. Der Speicherverbrauch ist ein Grund dafür, weshalb DOM Parser für sehr große XML Dateien nicht praktikabel sind. DOM Parser können, im Gegensatz zu SAX Parsern, auch HTML Dokumente[52] manipulieren.

  DOM ist ein vom W3C verabschiedeter Standard und liegt zur Zeit als DOM Level 2[53] vor.

  ”Die W3C-Spezifikation des Document Object Models (abgekürzt als: DOM) definiert eine Programmiersprachenunabhängig formulierte Menge abstrakter Schnittstellen zum lesenden und schreibenden Zugriff auf gültige HTML und wohlgeformte XML-Dokumente sowie eine Reihe weiterer Formate.” [Jec01c]

  Somit ist auch DOM, ähnlich zu SAX, ein API für den Zugriff auf XML Dokumente. Auf die weiteren Eigenschaften von DOM und der Bedeutung ans Objektmodell wird hier nicht näher eingegangen, da sie im Kontext von Web Services nicht von Bedeutung sind.

Um die Integration von XML in die Programmiersprache Java einfacher und objektorientierter zu gestalten, entwickelt SUN[54] die JAX APIs[55]. Sie sollen hier aufgezählt werden, da sie ebenfalls Bestandteil des von SUN propagierten ”Web Services Pack”[56] sein werden. Unter anderem gehören die folgenden APIs dazu (vgl. [Wol01] und [Hol01, S. 37 ff.]):

- **JAXP**
  Das Java API for XML Parsing ermöglicht einen abstrakten Zugriff auf verschiedene XML Parser. Der Parser kann dynamisch gewechselt werden, ohne den Anwendungscode ändern zu müssen. Dieses API ist bereits fertig und liegt als ”Final Release” vor.

- **JAXB**
  Die Java Architecture for XML Binding ermöglicht das automatische Abbilden von XML Dokumenten auf Java Objekte. Hierzu wird ein XML Schema analysiert und daraus die Java Klassen erstellt. Das API ist zur Zeit noch in der Entstehung, aber unter http://java.sun.com/xml/jaxb/ findet sich bereits eine ”Early Access” Implementation.

[52] HTML Dokumente sind in der Regel nicht XML-konform, da sie nicht wohlgeformt sind.
[53] DOM Level 3 liegt als Working Draft vor
[54] genauer der Java Community Process (JCP), http://www.jcp.org/
[55] http://java.sun.com/xml/javaxmlpack.html
[56] http://java.sun.com/j2ee/webservices/index.html

- **JAXM**
  Das Java API for XML Messaging definiert einen Standard für den Nachrichtenaustausch via XML. Das Nachrichtenformat und das Transportprotokoll sind zunächst offen gelassen, mögliche Formate sind aber z.B. SOAP oder ebXML. Es werden ebenfalls Ziele wie die garantierte Auslieferung einer Nachricht und die Definition erweiterter Sicherheitsmechanismen verfolgt. JAXM befindet sich zur Zeit im ”Public Review” Status .

- **JAXR**
  Das Java API for XML Registries bietet einen einheitlichen Zugriff auf verschiedene Verzeichnisdienste. Es ist sehr ähnlich zu JNDI, berücksichtigt aber spezielle XML Anforderungen. Von Anfang an sollen Zugriffe auf ebXML und UDDI Registries unterstützt werden. JAXR befindet sich zur Zeit im ”Public Review” Status .

- **Java API for XML-based RPC**
  Das API wird die framework-basierte Kommunikation zweier Systeme über Rechnergrenzen hinweg ermöglichen. Methodenaufrufe werden automatisch in XML umgewandelt und in dieser Form an die andere Anwendung geschickt. Hierbei steht besonders das Marshalling/Unmarshalling von Argumenten und die Definition der Abbildung von Java Methoden auf XML basierte RPCs im Vordergrund. Das API zur Zeit noch in der Entstehung.

- **Java API for WSDL (Nicht im JAX Pack enthalten)**
  Abstraktes API um WSDL Dateien zu bearbeiten. Das API ist zur Zeit noch in der Entstehung.

Sind diese APIs einmal fertiggestellt, so werden sie die Entwicklung von Web Services unter Java erheblich vereinfachen. Allerdings ist zur Zeit noch recht unklar, welche Rolle gerade JAXM und Java API for XML-based RPC innerhalb des Protokollstacks von Web Services spielen werden. Hier können gewisse Überschneidungen mit SOAP entstehen.

XML Namespaces
Da besonders SOAP und WSDL viel Gebrauch von XML Namensräumen machen, werden sie hier kurz behandelt (vgl. [Ma00, Kapitel 7]).

”An XML namespace is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names. XML namespaces differ from the ”namespace” conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set.” [BHL01]

XML Namensräume (*Namespaces*) sind Bezeichner, die an einen URI gebunden und dadurch innerhalb des definierten Kontextes eindeutig sind. Sie können sowohl in Verbindung mit Elementen wie Attributen verwendet werden. XML Namensräume unterscheiden sich von Namensräumen wie sie in Programmiersprachen (z.B. C++) oder in sonstiger Computerterminologie verwendet werden dadurch, dass sie eine interne Struktur beschreiben und keine mathematische Menge darstellen. XML Namensräume sind durch das W3C standardisiert worden. Sie erlauben einen modulareren Aufbau von XML Dokumenten und ein differenzierteres Parsing durch unterschiedliche Anwendungen. Weiterhin können Namenskonflikte vermieden werden, da gleichlautende lokale Bezeichner durch Zuweisung eines Namensraumes unterschieden werden können. Die Zuweisung eines Namensraumes gibt dem lokalen Bezeichner einen Kontext (vgl. [BHL01]).

XML Namensräume werden durch den Ausdruck `xmlns:myNamespace="Some-URI"`[57] definiert, der innerhalb eines Elements stehen muss. Ist diesem Element kein Namensraum Prefix vorangestellt, so hat es den Namensraum der innerhalb dieses Elements definiert worden ist. Namensraumbezeichner werden auf alle Kindelemente vererbt, können aber von diesen überschrieben werden. Es gibt einen Standard Namensraum (*Default-*

[57] Oft wird hier so etwas ähnliches wie `xmlns:myNamespace="http://example.org/mySpace"` verwendet. Dieser URI muss nicht notwendigerweise auf ein tatsächlich vorhandenes Ziel zeigen, er muss nur dokumentenweit eindeutig sein.

Namespace) der durch den Ausdruck `xmlns="Some-URI"` definiert wird und dann an alle Kindelemente vererbt wird, denen keine Namensraum Prefix vorangestellt ist. Der Standard Namensraum kann mit

```
xmlns=""
```

den betreffenden Bereich kein Namensraum definiert wäre (leerer Standard Namensraum).

```
<x xmlns:edi='http://ecommerce.org/schema'>
    <!--
        edi ist an http://ecommerce.org/schema gebunden,
        wenn es innerhalb von "x" Elementen und Inhalten
        verwendet wird.
     -->
</x>
```

Es wird zwischen ”qualifizierten Bezeichnern ” und ”unqualifizierten Bezeichnern ” unterschieden.

- **Qualifizierte Bezeichner (Qualified Names)**
  Diese Bezeichner setzen sich aus dem Namensraum Prefix und dem lokalen Element- bzw. Attributnamen zusammen[58]. An obiges Beispiel angelehnt, wäre ein qualifizierter Bezeichner z.B. `<edi:Header>`. Qualifizierte Bezeichner sind auch Elementnamen, denen keine Namensraum Prefix vorangestellt ist, die aber den Standard Namensraum erben (falls vorhanden). Attribute die innerhalb eines Elements mit einem qualifizierten Bezeichner liegen, sind immer qualifiziert.
- **Unqualifizierte Bezeichner (Unqualified Names)**
  Unqualifizierte Bezeichner sind Element- oder Attributnamen, denen kein Namensraum Prefix vorangestellt ist und die auch keinen Namensraum erben. Attribute ohne Namensraum Prefix haben den gleichen Namensraum wie das Element, in dem sie liegen. Ist dieses Element unqualifiziert so ist es auch das Attribut.

XML Schema
Der W3C-Standard XML-Schema (XSD) ist eine XML-Sprache zur Definition von XML-Vokabularen und der Nachfolger der Document Type Definitions (DTD)[59]. Er wurde vom W3C im Mai 2001 als Standard verabschiedet. Die DTD wurde bisher verwendet, um den Aufbau und die Struktur eines XML Dokuments zu definieren. Im Gegensatz zur DTD ist XML Schema jedoch daten- statt dokumentenorientiert. XML Schema definiert eine Klasse von XML Dokumenten. Darin werden sowohl die Struktur wie auch die verwendeten Datentypen spezifiziert. Genügt ein XML Dokument einem Schema, so spricht man von einer Dokumenten-Instanz. Aber ein XML Dokument muss nicht notwendigerweise einem XML Schema oder einer DTD genügen. Schemas selbst sind ebenfalls XML Dokumenten-Instanzen, die der Schema Definition für XML Schemas genügen [60]. XML Schema erweitert die Ausdrucksmöglichkeiten der DTD hinsichtlich der Definition von XML Dokumenten. Dies gilt einerseits in Bezug auf inhaltliche Merkmale wie Datentypen und Wertebereiche sowie andererseits in Bezug auf strukturelle Merkmale. Durch die stärkere Typisierung der Daten ist es möglich, schon zum Erstellungszeitpunkt einer XML Dokumenten-Instanz ungültige Werte zu erkennen. Dies garantiert, hinsichtlich einer maschinellen Verarbeitung, Stabilität und Wahrung der Datenintegrität zur Laufzeit. Weiterhin unterstützt XSD im Gegensatz zur DTD die Spezifizierung von Auftrittshäufigkeiten bestimmter Elemente, die Wiederverwendbarkeit von Attributdeklarationen, eine Erweiterung den Typsystems durch den Anwender und den Einsatz von Namensräumen.

XML Schema wird im Rahmen dieser Arbeit nur kurz betrachtet, da ein tieferer Einblick für das Verständnis von Web Services nicht notwendig ist. Der interessierte Leser sei hier auf [W3C01b] und [Cos01] verwiesen.

[58] durch ”:” getrennt
[59] Die DTD ist nicht Bestandteil dieser Arbeit.
[60] http://www.w3.org/TR/xmlschema-1/
http://www.w3.org/TR/xmlschema-2/

Mit XML Schema kann die Struktur eines XML Dokuments definiert werden. Dies umfasst die zulässigen Elemente und Attribute sowie deren Beziehungen untereinander. Für Elemente können ihre zulässigen Inhalte und für Attribute ihre zulässigen Werte festgelegt werden. Es kann der Datentyp und der Wertebereich angegeben werden. Ferner lassen sich die Auftrittshäufigkeiten einzelner Elemente und Attribute definieren.

Das folgende XML Dokument enthält eine Auflistung von Personen, die jeweils vier Eigenschaften besitzen. Eine Person wird durch ihren Vornamen, Nachnamen, ihre Anrede und ihr Alter beschrieben. Die Anrede soll als Attribut zu dem Element `Nachname` angegeben werden. Es muss mindestens eine Person existieren. Bis auf das Alter, welches ganzzahlig und positiv ist, sind alle anderen Datentypen Zeichenketten.

```
<?xml version="1.0"?>
<Personen>
    <Person>
        <Vorname>James Tiberius</Vorname>
        <Nachname Anrede="Captain">Kirk</Nachname>
        <Alter>40</Alter>
    </Person>
    <Person>
       <Vorname>Leonard</Vorname>
       <Nachname Anrede="Doktor">McCoy</Nachname>
       <Alter>51</Alter>
    </Person>
</Personen>
```

Die XML Schema Definition für das oben stehende XML Dokument sieht wie folgt aus:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">\

<xsd:complexType name="PersonenType">
    <xsd:sequence>
        <xsd:element name="Person"
                     type="PersonType"
                     minOccurs="1"
                     maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="Personen" type="PersonenType"/>

<xsd:complexType name="PersonType">
    <xsd:sequence>
        <xsd:element name="Vorname"  type="xsd:string"/>
        <xsd:element name="Nachname" type="NachnameType"/>
        <xsd:element name="Alter"    type="xsd:short"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="NachnameType">
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute name="Anrede" use="required"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

</xsd:schema>
```

In der ersten Zeile wird der Namensraum `xsd` definiert und an den URI für die XML Schemadefinition gebunden. Dies soll helfen zu erkennen, ob die in dem Dokument referenzierte Objekte zur Schemadefinition gehören oder vom Autor selbst geschaffene Objekte sind.

Darauf wird ein komplexer (zusammengesetzter) Datentyp mit dem Namen `PersonenType` definiert. Dieser besteht nur aus Wiederholungen (`xsd:sequence`) des Elements `Person`, welches vom Datentyp `PersonType` ist. Das Element `Person` muss mindestens einmal (`minOccurs="1"`), darf aber unendlich oft (`maxOccurs="unbounded"`) vorkommen.

Danach wird das Wurzelelement `Personen` definiert welches vom oben definierten Typ `PersonenType ist`.

Der zusammengesetzte Datentyp `PersonType` wird als Abfolge der Elemente `Vorname, Nachname und Alter` festgelegt. Der Vorname ist eine Zeichenkette (`xsd:string`), der Nachname vom `Typ NachnameTyp` und das Alter eine positive ganze Zahl.

Nun fehlt noch der Datentyp `NachnameType`, der hier als letztes definiert wird. Normalerweise ist der Nachname lediglich eine Zeichenkette. Da hier aber noch das Attribut `Anrede` eingebettet sein soll, muss der Nachname als zusammengesetzter Datentyp realisiert werden. Das Element `xsd:simpleContent` sagt aus, dass sich innerhalb des Nachname Elementes keine weiteren Elemente befinden dürfen. Der zusammengesetzte Datentyp `NachnameType` wird duch `xsd:extension` um ein Attribut mit dem Namen Anrede erweitert. Innerhalb des `xsd:extension` Elements wird der Datentyp des Attributes (hier Zeichenkette) angegeben. Das Attribut muss zwingend vorhanden sein (`required`).

4.4.3 SOAP für entfernte Methodenaufrufe

SOAP stand bis Version 1.1 inklusive für "Simple Object Access Protocol". Übersetzt würde dies folgendes bedeuten: Einfaches Protokoll für den Zugang zu Objekten. Da SOAP aber eigentlich weder etwas mit Objekten oder Objektorientierung noch mit dem Zugriff auf Objekte zu tun hat, bedeutet das Akronym SOAP ab Version 1.2 nur noch "SOAP". Die von IBM getroffene Aussage, SOAP würde irgendwann durch XML Protocol ersetzt werden, ist irreführend. Laut Mario Jeckle[61] (Mitglied der XML-Protocol Working Group) referenziert XML-Protocol (wenn es als Protokoll gemeint ist) SOAP 1.2. Korrekterweise ist die Bezeichnung XML-Protocol der Name der Working Group des W3C für SOAP 1.2. Dies ist auch die Version, auf die sich diese Arbeit bezieht. Ferner wird hier nur die RPC-Semantik von SOAP betrachtet, da für Web Services der Austausch von Nachrichten, die keinen RPC repräsentieren, praktisch bedeutungslos ist. Die über SOAP transportierten Informationen können aber prinzipiell beliebiger Gestalt sein, solange es XML Dokumente sind [CCVC01, S. 106].

"SOAP version 1.2 provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding application defined data. This allows SOAP to be used in a large variety of systems ranging from messaging systems to remote procedure calls (RPC)." [GHMN01]

SOAP stellt einen einfachen und leichtgewichtigen Mechanismus zur Verfügung, der den Austausch von strukturierten oder typenorientierten Daten in einer dezentralen und verteilten Umgebung unter Verwendung von XML erlaubt. SOAP selbst beschreibt keine Geschäftslogik, sondern ist ein Mechanismus, um Zugang zu einer Anwendung zu erlangen. Dies geschieht zum einen in der Form eines modularen Modells zum Verpacken anwendungsspezifischer Daten und zum anderen existiert ein Kodierungsmechanismus für anwendungsspezifische Daten.

Weyer (vgl. [Wey01a]) definiert SOAP wie folgt: "Mechanismus zum Austausch von strukturierter und typisierter Information zwischen Kommunikationsendpunkten auf Basis von XML."

[61] in einem persönlichen Gespräch am 31.10.2001 auf der Web Service Konferenz in Stuttgart

”The most fundamental concept of the SOAP model is the use of XML documents as messages. SOAP messages are XML.” [CCVC01, S. 79] Das grundlegende Konzept von SOAP ist, dass XML Dokumente Botschaften sind.

SOAP wurde vor allem unter der Maßgabe der Einfachheit und Erweiterbarkeit geschaffen und besitzt daher kein verteiltes Speichermanagement, erlaubt keine multiplen RPCs innerhalb einer Nachricht, keine Stapelverarbeitung von Nachrichten und keine Übergabe von Referenzen auf ein Objekt. Weiterhin wird auch die Objekt-Aktivierung[62] nicht unterstützt. Es werden keine Aussagen über Typsicherheit, Versionierung und bidirektionale HTTP Kommunikation gemacht (persistente HTTP Verbindungen). Vergleiche hierzu [GHMN01] und [Mod01].

Mit Hilfe von SOAP ist es möglich, standardisierte Nachrichten zwischen Anwendungen auszutauschen. Diese Nachrichten können entweder Methoden-/Funktionsaufrufe oder Dokumente enthalten[63]. Die bisher zur Verfügung stehenden Mechanismen wie CORBA, RMI und DCOM sind jeweils nur in einem eingeschränkten Kreis verwendbar und berücksichtigen nicht die Besonderheiten des Internets. Werden SOAP Nachrichten über HTTP übertragen, so existiert, im Gegensatz zu den anderen Protokollen, keine Firewall Problematik, da HTTP normalerweise keinen Restriktionen unterliegt. SOAP ist eine Kombination von standardisierten XML Nachrichten und Standard Internet Protokollen und verwendet damit ausschließlich die in der Industrie akzeptierten Standards. SOAP löst das Problem, Anwendungen innerhalb einer heterogenen Systemlandschaft plattformübergreifend miteinander kommunizieren zu lassen. Dies geschieht im wesentlichen dadurch, existierende Programme für andere nutzbar zu machen und Software damit als Dienstleistung zu sehen. Dabei ist besonders hervorzuheben, dass hier keine neue Technologie erfunden worden ist, sondern die bereits bestehenden und bewährten Technologien kombiniert und neu angewendet wurden. Wie dies im Detail aussieht, ist Thema dieses Kapitels.

Die folgende Beschreibung lehnt sich an [GHMN01] an: Ein SOAP Nachricht besteht aus vier Teilen:

1. **dem SOAP Umschlag (Envelope)**
   Der SOAP Umschlag umschließt alle anderen Teile einer SOAP Nachricht. Er gibt Auskunft darüber, was sich in der Nachricht befindet, für wen sie bestimmt ist und welche Teile optional sind.
2. **dem SOAP Binding Framework**
   Eine abstrakte Beschreibung, über welches darunterliegende Transportprotokoll der SOAP Umschlag transportiert werden soll. SOAP ist vom Transportprotokoll unabhängig. Es ist z.B. eine Übertragung über HTTP, SMTP, FTP, E-Mail, MQ[64], IIOP oder Fax denkbar. In dieser Arbeit wird nur auf die Bindung an HTTP eingegangen.
3. **den SOAP Kodierungsvorschriften (Encoding Rules)**
   Hier werden Serialisierungsmechanismen definiert, um anwendungsspezifische Datentypen austauschen zu können.
4. **der SOAP RPC (Remote Procedure Call) Darstellung**
   Definiert eine Konvention um RPC Aufrufe zu ermöglichen.

Darüber hinaus wird ein Modell für den Austausch von SOAP Nachrichten definiert (vgl. [CCVC01, S. 78]):

1. **Alle Nachrichten sind XML Dokumente.**
2. **Eine Nachricht wird immer von einem Sender an einen Empfänger geschickt.**
3. **Nachrichten können über Zwischenknoten geleitet werden.**

[62] Aktivierung von ”schlafenden” Objekten, vgl. http://java.sun.com/products/jdk/1.2/docs/guide/rmi/activation.html

[63] Der Unterschied dieser beiden ”Betriebsarten” ist der, dass im Gegensatz zum RPC Modus beim Nachrichten/Dokumenten Modus keine Antwort generiert wird.

[64] Message Queues, z.B. MQSeries von IBM

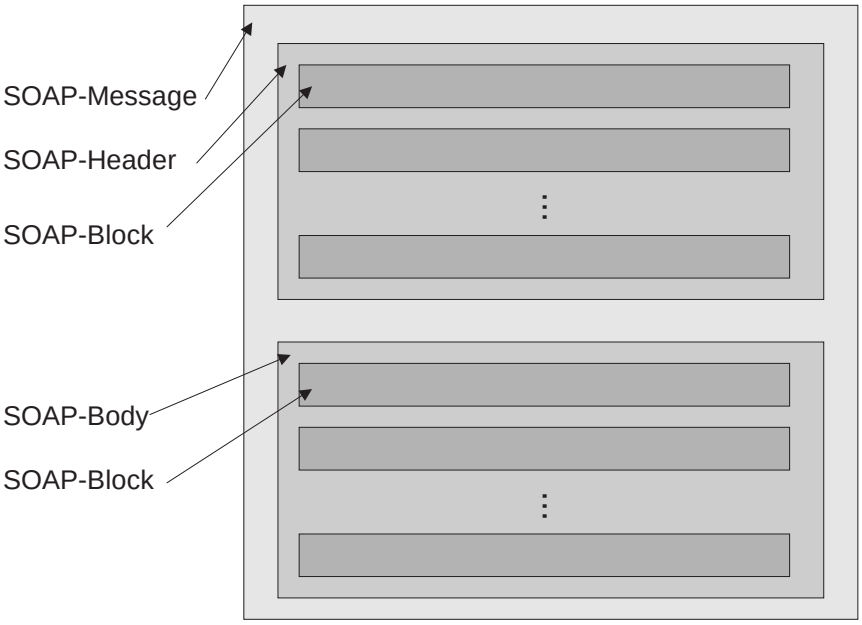

Abbildung 13: Aufbau einer SOAP Nachricht

Eine SOAP Nachricht kann z.B. so aussehen:

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: localhost:8080
Content-Type: text/xml;charset=utf-8
Content-Length: 620
SoapAction: ""

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">

    <SOAP-ENV:Body>
        <ns1:getMD5
            xmlns:ns1="urn:MD5"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

        <toEncrypt
            xsi:type="xsd:string"
            SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/">Hallo Welt</toEncrypt>

        </ns1:getMD5>
    </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Dieser SOAP Umschlag dient dazu, eine Methode `getMD5` aufzurufen, die als Parameter einen String erwartet und als Rückgabewert einen String zurückgibt. Demnach sieht die Signatur folgendermaßen aus: `String getMD5(String encrypt)`. `getMD5` errechnet den MD5 Hash über eine gegebene Zeichenkette.

Die einzelnen Teile dieser SOAP Nachricht sollen im folgenden besprochen werden:

Die ersten fünf Zeilen sind der HTTP Header (in diesem Beispiel wird die SOAP Nachricht über HTTP übertragen). Dieser gehört nicht zur SOAP Nachricht, sondern ist Bestandteil von HTTP. Der `Content-Type` im HTTP Header muss `text/xml` sein und die eigentliche SOAP Nachricht ist im HTTP Body kodiert. Wird SOAP über HTTP übertragen, so muss ein zusätzlicher HTTP Header `SOAPAction` angegeben werden. Dieser kennzeichnet die HTTP Nachricht als SOAP Nachricht und kann weitere Informationen über den eigentlichen Inhalt bereitstellen. Er wird vom HTTP Server oder von Zwischensystemen (z.B. Firewalls) ausgewertet.

```
<?xml version='1.0' encoding='UTF-8'?>
```

Kennzeichnet das folgende Dokument als XML 1.0 Dokument.

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
```

Hier beginnt der Umschlag. Es werden drei XML Namensräume definiert.

- **xmlns:SOAP-ENV**
- **xmlns:xsi**
- **xmlns:xsd**

```
<SOAP-ENV:Body>
        <ns1:getMD5
            xmlns:ns1="urn:MD5"
            SOAP-ENV:encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/">
```

Hier beginnt der Body der SOAP Nachricht[65], der Teil, der die Nutzdaten enthält. Zu Beginn wird der Name der Methode genannt, die aufgerufen werden soll. In diesem Falle ist es `getMD5`. Es wird ein weiterer Namensraum `ns1` definiert und mit dem Namen des aufzurufenden Dienstes am Kommunikationsendpunkt, verknüpft. Hier heißt dieser Dienst *MD5*. Die nächste Zeile legt die Kodierung der Datentypen auf die SOAP Standardkodierung fest.

```
<toEncrypt
            xsi:type="xsd:string"
            SOAP-ENV:encodingStyle=
       "http://schemas.xmlsoap.org/soap/encoding/">Hallo Welt</toEncrypt>

        </ns1:getMD5>
    </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Da die Methode `getMD5` einen String als Eingabeparameter erwartet, muss dieser mitübergeben werden. Der Inhalt des `<toEncrypt>` Elementes wird durch eine XML Schema Definition als String gekennzeichnet und aufgrund seiner Position innerhalb des Methodenaufrufs als Übergabeparameter interpretiert. Es folgen noch die jeweiligen Endtags des Body, und des Umschlages und die SOAP Nachricht ist komplett. Sie wird nun mittels HTTP POST an den Server (Kommunikationsendpunkt) geschickt und dort ausgewertet.

[65] Diese SOAP Nachricht hat keinen Header. Dieser ist optional.

In einem SOAP Umschlag kann sich aber neben dem Body, der die Nutzdaten enthält, auch noch ein Header befinden. Dieser ist optional und trägt ggf. Metainformationen. Dies können u.a. Informationen über Session-IDs, Verschlüsselung, Zwischenknoten und Transaktionen sein. Aber auch anwendungsspezifische Informationen können frei definiert und im Header übertragen werden. Da der SOAP Header zur Zeit wenig praktische Bedeutung hat, wird an dieser Stelle nicht näher darauf eingegangen.

Tritt während eines SOAP Nachrichtenaustauschs ein Fehler auf, so wird dies durch das *Fault-Element* gekennzeichnet. Die SOAP Antwort enthält dann ein Element mit folgender Struktur:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultcode>SOAP-ENV:Server</faultcode>
            <faultstring>service 'urn:MD5' unknown</faultstring>
            <faultactor>/soap/servlet/rpcrouter</faultactor>
        </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Das `faultcode` Element spezifiziert die Fehlerklasse oder teilt mit, wo der Fehler aufgetreten ist. Eine ausführliche Fehlerbeschreibung findet sich im `faultstring` Element. Das `faultfactor` Element spezifiziert den Endpunkt, an dem der Fehler aufgetreten ist. Diese Information ist vor allen Dingen beim Einsatz von Zwischenknoten wichtig, da es hier weitere Fehlerquellen als nur den Sender und Empfänger geben kann.

Damit ein Server SOAP Nachrichten verstehen kann, muss dort eine SOAP Implementierung installiert sein. Diese hat die Aufgabe, die eingehende SOAP Nachricht auf ihre Gültigkeit zu prüfen und auszuwerten. Diese Auswertung besteht im wesentlichen darin, die gewünschte Methode auf dem Server auszuführen und das Ergebnis an den Aufrufer zurückzusenden.

Zur Zeit sind u.a. folgende SOAP Implementierung verfügbar (vgl. [Wey01a]):

- **Apache SOAP**
  SOAP 1.1[66] Implementierung in Java. Apache SOAP entstand aus dem SOAP4J Projekt von IBM, welches an Apache übergeben wurde. http://xml.apache.org/soap/
- **MS SOAP Toolkit**
  http://msdn.microsoft.com/soap/
- **MS .NET Runtime**
- **IONA Orbix**
  http://www.iona.com/
- **pocketSOAP**
  http://www.pocketsoap.com/
- **Diverse Implementierungen**
  http://www.soap-wrc.com/

Im folgenden wird nun exemplarisch eine SOAP Implementierung näher betrachtet. Aufgrund der weiten Verbreitung und der offenen Quellen wird die Apache SOAP Implementierung gewählt.

[66] Eine Features der Version 1.1 werden noch nicht unterstützt.

Apache SOAP Implementierung
Apache SOAP[67] ist frei erhältlich und wird unter der Apache Software License[68] vertrieben. Der HTTP Kommunikationsendpunkt wird durch ein Servlet mit Name *rpcrouter* für RPC orientierte Nachrichten bzw. *messagerouter* für Message orientierte Nachrichten realisiert. Um mittels SOAP einen RPC ausführen zu können, muss zuerst die Anwendung, deren Methoden exportiert werden sollen, registriert werden. Diese Anwendung kann entweder eine Java Klasse, eine EJB, oder ein BSF[69] Skript sein. Die Registrierung dieser Anwendung geschieht mittels eines Deployment Descriptors, oder einer webbasierten Oberfläche. Dort müssen, für eine Java Klasse, folgende Daten angegeben werden:

- **service-urn**
  Dies ist der eindeutige Bezeichner (URN, Uniform Resource Name) für den zur Verfügung gestellten Dienst. In dem hier besprochenen Beispiel ist der service-urn `urn:MD5`.
- **exposed-methods**
  Durch Leerzeichen getrennte Aufzählung der Methoden, die exportiert werden sollen. Im MD5 Beispiel wird lediglich eine Methode mit dem Namen `getMD5` exportiert.
- **implementing-class**
  Vollständiger Klassenname (mit Packagenamen) der Klasse, die die Service Implementierung und die zu exportierenden Methoden bereitstellt.
- **type (optional) Attribute**
  Ist type=”message” gesetzt, so handelt es sich nicht um einen RPC Service, sondern um einen dokumentenorientierten Service.
- **checkMustUnderstands (optional)**
  Gibt an, ob das `MustUnderstand` Feld des SOAP Headers geprüft werden soll.
- **scope**
  Der Scope des Services legt den Lebenszyklus der Instanz der ”implementing-class” fest. Er muss einer der drei folgenden Werte besitzen:
  - **Request**
    Das Objekt lebt nur solange, bis die Anfrage beantwortet wurde.
  - **Session**
    Das Objekt lebt für die Dauer einer HTTP Session.
  - **Application**
    Das Objekt hat die gleiche Lebensdauer wie das Servlet, welches den Aufruft bedient. Das heißt, dass unterschiedliche Sessions durch dasselbe Objekt bedient werden. Dies bringt zwar deutlich Geschwindigkeitsvorteile, aber es muss auf mögliche Sicherheitsrisiken geachtet werden.
- **static (optional)**
  Gibt an, ob die exportierten Methoden statische (also Klassen-) Methoden sind.

Der Deployment Descriptor für den MD5 Service sieht demnach folgendermaßen aus:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="urn:MD5">
  <isd:provider type="java"
                scope="Application"
                methods="getMD5">
```

[67] http://xml.apache.org/soap/docs/index.html
[68] Open Source Lizenz, d.h. der Quelltext liegt offen.
[69] Das Bean Scripting Framework ist eine Architektur, um Skriptsprachen mit der Java Welt zu verbinden. Nähere Informationen unter http://www-124.ibm.com/developerworks/projects/bsf

```
    <isd:java class="MD5Calculator" static="false"/>
  </isd:provider>
  <isd:faultListener>
    org.apache.soap.server.DOMFaultListener
  </isd:faultListener>
</isd:service>
```

Zum Schluss wird noch ein *Fault Listener* definiert, der die Fehlerbehandlung übernimmt, falls Fehler auftreten. `DOMFaultListener` ist der Standard *Fault Listener* der Apache SOAP Implementierung.

Nun wird der Deployment Descriptor in einer Datei mit Namen DeploymentDescriptor.xml abgespeichert. Danach kann der Service mit folgendem Aufruf registriert (*deployed*) werden:

```
java org.apache.soap.server.ServiceManagerClient
     http://localhost:8080/soap/servlet/rpcrouter deploy DeploymentDescriptor.xml
```

Die Apache SOAP Implementierung weist noch ein paar Unzulänglichkeiten auf. Unter anderem kann es XML Schema Konflikte geben, da zur Zeit drei Versionen von Schemata existieren. Ferner werden einige Feature wie z.B. der `mustUnderstand` Parameter noch nicht unterstützt. Eine vollständige Liste findet sich unter [Apa01].

Die SOAP Antwort aus obigem Beispiel sieht wie folgt aus:

```
HTTP/1.0 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 473

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema">

    <SOAP-ENV:Body>
        <ns1:getMD5Response xmlns:ns1="urn:MD5"
                            SOAP-ENV:encodingStyle=
                            "http://schemas.xmlsoap.org/soap/encoding/"
 >

            <return xsi:type="xsd:string">
                5c372a32c9ae748a4c040ebadc51a829
            </return>
        </ns1:getMD5Response>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In diesem relativ einfachen Beispiel wurden lediglich Zeichenketten als Datentyp ausgetauscht. Für "Real-World" Anwendungen reichen primitive Datentypen wie Integer, Float, String oder Boolean aber nicht aus. Es ist eine weitere Aufgabe von SOAP, sowohl primitive wie komplexe Datentypen in XML und wieder zurück umzuwandeln. Diesen Vorgang nennt man Marshalling/Unmarshalling. Hierbei spielt die Kodierung (*encoding*) und das Abbilden von Datentypen auf XML (*type mapping*) eine wichtige Rolle.

Encoding
Unter Encoding versteht wird die Art und Weise verstanden, wie bestimmte Datentypen in XML dargestellt werden. Die Apache SOAP Implementierung unterstützt die Standardtypen von SOAP 1.1, die in dem Schema

http://schemas.xmlsoap.org/soap/encoding/ definiert sind, sowie den Datentyp org.w3c.dom.Element und die XMI (XML Metadata Interchange) Kodierung. Darüber hinaus bietet Apache SOAP eine ganze Reihe Serialisierer für häufig verwendete Datentypen in Java. Unter anderem sind dies alle primitiven Datentypen, Arrays, Hashtable, Date, InputStream, Java Beans, u.v.m. .Besonders der Java Beans Serialisierer ist sehr interessant, da er beliebige Java Objekte, die der Java Bean Definition genügen[70], in XML darstellen kann. Für jede zu serialisierende Bean muss dies explizit im *Deployment Descriptor* angegeben werden. Dies nennt man *Type Mapping*, da ein Java Datentyp (Bean Klasse) auf XML abgebildet wird. Der relevante Abschnitt im *Deployment Descriptor* sieht (in Anlehnung an [Apa01]) wie folgt aus:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="...">
  <isd:provider .../>
  <isd:faultListener .../>
  <isd:mappings [defaultRegistryClass="registry-class"]>
    <isd:map encodingStyle="encoding-uri"
             xmlns:x="qname-namespace"
             qname="x:qname-element"

             javaType="java-type"
             java2XMLClassName="serializer"
             xml2JavaClassName="deserializer"/>
    ...
  </isd:mappings>
...
</isd:service>
```

Die interessanten Werte sind hier

- **encoding-uri**
  Spezifiziert die Kodierungsvorschrift (typischerweise http://schemas.xmlsoap.org/soap/encoding).
- **qname-namespace**
  Namensraum für das XML Element.
- **qname-element**
  Name des XML Elementes.
- **java-type**
  Die Java Klasse die serialisiert/deserialisiert werden soll. Wird der Bean Serialisierer bzw. Deserialisierer verwendet, muss diese Java Klasse eine Java Bean sein.
- **serializer/deserializer**
  Der Klassenname des Serialisierers/Deserialisierers. Für Java Beans ist dies:
  org.apache.soap.encoding.soapenc.BeanSerializer.

Für alle anderen Java Datentypen müssen eigene Serialisierer/Deserialisierer geschrieben werden.

SOAP API

Um mittels eigener Anwendungen SOAP Nachrichten verarbeiten zu können, stellt die Apache SOAP Implementierung ein API zur Verfügung. Dieses API findet sich unter http://xml.apache.org/soap/docs/apiDocs/ und soll hier kurz erläutert werden. Die Beschreibung orientiert sich an [Apa01]. Um einen SOAP Client zu schreiben, der einen SOAP RPC ausführt, sind folgende Schritte notwendig.

[70] http://java.sun.com/products/javabeans/docs/spec.html

1. Zunächst muss die Signatur der Methode, die man per RPC aufrufen möchte, in Erfahrung gebracht werden. Diese Information kann der WSDL Datei entnommen werden.

   In unserem Fall ist die Signatur: `String getMD5(String toEncrypt)`

2. Dann muss sichergestellt sein, dass für jeden Datentyp, der innerhalb des RPC übertragen werden muss, Serialisierer bzw. Deserialisierer existieren. Ist dies nicht der Fall, müssen diese geschrieben und registriert werden.

   Der MD5 Service überträgt nur Zeichenketten. Dieser Datentyp ist bereits durch die SOAP Implementierung abgedeckt

3. Jetzt kann damit begonnen werden, den Java Code für den SOAP Client zu schreiben. Zentraler Bestandteil ist hierfür das `Call` Objekt, welches zuerst instanziiert werden muss.

```
org.apache.soap.rpc.RPCMessage.Call call =
       new org.apache.soap.rpc.RPCMessage.Call();
```

4. Dem `Call` Objekt muss mitgeteilt werden, wie der Service heißt, für den dieser RPC bestimmt ist. Dies geschieht mittels der `setTargetObjectURI()` Methode.

```
Im Falle des MD5 Services sieht dies wie folgt aus:
        call.setTargetObjectURI(''urn-MD5'');}
```

5. Danach muss dem `Call` Objekt der Name der Methode, welche auf dem Server aufgerufen werden soll, bekannt gemacht werden. Die Methode muss im *Deployment Descriptor* exportiert worden sein.

```
call.setMethodName(”getMD5”);
```

6. Falls die Methode Parameter erwartet, müssen diese mit `call.setParams()` registriert werden. Für die Parameter ist ebenfalls Punkt 2 zu beachten.

```
String toEnryptValue = ''Hallo Welt'';
Vector params = new Vector();
   params.addElement(new Parameter(''toEncrypt'',String.class,
                            toEnryptValue,Constants.NS_URI_SOAP_ENC));
   call.setParams(params);
```

7. Nun wird die SOAP Nachricht abgeschickt, indem die `invoke()` Methode auf dem Call Objekt aufgerufen wird. Diese Methode hat einen Rückgabewert vom Typ `Response` und dieser beinhaltet die Antwort auf die SOAP Nachricht. Die `invoke()` Methode hat zwei Parameter. Der erste ist der URI des Kommunikationsendpunktes für RPC Nachrichten des SOAP Servers. Der zweite ist der Wert des SOAPAction Parameters im HTTP Header.

```
Response response = call.invoke("http://localhost:8080/
 soap/servlet/rpcrouter","");
```

8. Nachdem die Antwort empfangen wurde (da der RPC Aufruf synchron ist, kann dies einige Zeit dauern), muss geprüft werden, ob ein Fehler aufgetreten ist. Dies geschieht mittels der `generatedFault()` Methode.

```
if (!response.generatedFault()){
... kein Fehler
}else{
... Fehler
}
```

9. Wurde ein Fehler festgestellt, so kann der Fehler mit `getFault()` ausgewertet werden. War der Aufruf erfolgreich, so kann mit der Methode `getReturnValue()` der Rückgabewert des RPC ermittelt werden.

```
if (!response.generatedFault()){
Parameter result = response.getReturnValue();
System.out.println((String) result.getValue());
}else{
Fault fault = response.getFault();
System.out.println("SOAP-Fehler: "+fault.getFaultString());
}
```

Nachteile von SOAP
Durch die Verwendung von XML entsteht in erster Linie ein Performanz Problem, da ein reines Text Protokoll immer langsamer sein wird als ein binäres Protokoll. Insbesondere das Parsen und Auswerten der XML Daten ist rechenintensiv. Dies ist vor allen Dingen serverseitig ein Problem, da hier unter Umständen mehrere hundert Anfragen gleichzeitig bedient werden müssen. Clientseitig spielt der Geschwindigkeitsaspekt praktisch keine Rolle. Des weiteren sind XML kodierte Daten relativ umfangreich, was eine höhere Datenmenge zur Folge hat. Es müssen in XML neben den eigentlichen Nutzdaten auch die gesamten Informationen zu Struktur und Aufbau mitübertragen werden. Die Folge ist eine erhöhte Netzwerkbelastung. XML Daten können zwar gut komprimiert werden, aber die Kompression bzw. Dekompression führt besonders serverseitig zusätzlich zu einer Erhöhung der Rechenzeit. Hier muss also zwischen Netzwerkbelastung und Serverauslastung abgewogen werden. Zwei weitere Probleme, die sich mit der Verwendung von SOAP stellen, sind die bisher unzureichende Standardisierung und die Firewall Durchlässigkeit. Letztere ist gleichzeitig ein Vorteil, kann jedoch ernste Sicherheitsprobleme verursachen. Abhilfe kann hier durch inhaltsbasierte Filterung (z.B. Erkennung von `SOAPAction` im HTTP Header) oder entsprechende Authentifizierungsmechanismen geschaffen werden. Dieser Punkt wird in Kapitel 4.5.1 ausführlich behandelt.

SOAP Erweiterungen
Ab SOAP Version 1.2 gibt es eine Reihe von Erweiterungen (*Extensions*), die je nach Bedarf als ”Plug-In” hinzugefügt werden können. Sie werden durch Manipulation des SOAP Headers realisiert. Allerdings befinden sich die SOAP Erweiterungen momentan noch im Entwicklungsstadium, weshalb hier nur kurz darauf eingegangen werden soll. Es existieren bisher kaum Implementationen, die die Erweiterungen unterstützen (vgl. [CCVC01, S. 23 ff.]).

Die SOAP Erweiterungen:

- **Attachments**
  Bietet die Möglichkeit zum Transport von binären Daten innerhalb einer SOAP Nachricht. Die binären Daten werden hierzu MIME und Base64 kodiert.
  Nähere Informationen sind unter http://www.w3.org/TR/SOAP-attachments zu finden.

- **Routing/Zwischenknoten**
  Eine Nachricht kann auf dem Weg vom Sender zum Empfänger über Zwischenknoten geleitet werden. Diese nehmen an der Nachricht gewisse Modifikationen vor und schicken sie weiter. Alle notwendigen Routing-Informationen und Anweisungen für die einzelnen Zwischenknoten stehen im SOAP Header. Dies ermöglicht z.B. die Aggregation von Web Services. Es ist weiterhin denkbar, einen solchen Mechanismus zu verwenden um ausfallsichere, performante oder skalierbare Systeme zu schaffen. Dies kann u.a. durch Speicherung von Informationen in den Zwischenknoten (*caching* oder *store-and-forward*), alternatives Routing bei ausgefallenen Knoten oder durch verteilte Abarbeitung der Anfrage realisiert werden.

- **Zuverlässiger Transport**
  Muss auf Anwendungsschicht programmiert werden. Notwendige Daten hierfür werden im SOAP Header übertragen. Dieses Thema wird in Kapitel 4.5.2 näher beschrieben.

- **Sicherheit**
  Vgl. Kapitel 4.5.1
- **Quality of Service**
  Vgl. Kapitel 4.5.2
- **Kontextsensitivität**
  Kontextsensitiv wird oft auch mit ”intelligent” gleichgesetzt. Vgl. auch Kapitel 4.5.4
- **Transaktionen**
  Vgl. Kapitel 4.5.3

Zukunft von SOAP
Zur Zeit arbeitet das W3C an der Standardisierung von SOAP 1.2. Apache arbeitet schon an der Nachfolgeimplementation von Apache SOAP mit dem Namen Axis. Axis soll SOAP 1.2 vollständig unterstützen und darüber hinaus weitere Tools (z.B. Generierung von WSDL Dateien) zur Verfügung stellen. Axis ist durch die Verwendung von SAX (statt DOM) wesentlich performanter, ist aber zur Zeit noch im Alpha Stadium. Aus architektonischer Sicht stellt sich die Frage, ob sich SOAP eher als Client- oder Backendprotokoll behaupten wird. Als Backendprotokoll hätte es die Aufgabe, die dort vorherrschenden Technologien wie z.B. CORBA oder IIOP abzulösen. Einen Vorteil, der gewonnen würde, wäre Interoperabilität im Backenendbereich. Dies müsste mit Nachteilen, was die Performanz betrifft, erkauft werden. Microsoft stuft SOAP als Clientprotokoll ein, aber wie und wo SOAP sich letztendlich durchsetzen wird, bleibt abzuwarten. SOAP wird wohl eher die Stelle eines Mediators zwischen den heterogenen Welten CORBA, DCOM und EJB einnehmen.

4.4.4 WSDL zur Beschreibung und Bindung

”WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.” [CCMW01]

Die WSDL ((Web Service Description Language)) ist eine auf XML basierende Sprache, um Kommunikationsendpunkte zu beschreiben. Diese Endpunkte können Nachrichten verarbeiten, die entweder dokumenten- oder prozedurorientiert (im Sinne eines RPCs) sind. Die Operationen und Nachrichten werden zunächst abstrakt beschrieben und danach konkretisiert. WSDL ist flexibel, da die Endpunkte und Nachrichten unabhängig von der konkreten Ausprägung der Transportprotokolle und Nachrichtenformate, beschrieben werden können. Es wird hier nur die konkrete Bindung an das SOAP und MIME Nachrichtenformat und an das HTTP Transportprotokoll beschrieben.

WSDL ist nach [Wey01a] eine Zusammenführung von IBMs NASSL[71] und Microsofts SDL/SCL[72] und liegt zur Zeit als W3C Note in der Version 1.1 vor. Eine ”W3C Note” ist lediglich ein an das W3C übergebener Vorschlag, der geprüft, bearbeitet und bewertet wird. Falls das Ergebnis den Anforderungen des W3Cs entspricht, wird das Dokument als Standard verabschiedet. Somit ist WSDL noch kein offizieller Standard. Dies erklärt auch die bisher noch existierenden Inkompatibilitäten der verschiedenen Toolkits zur Erzeugung und Bearbeitung von WSDL Dateien. Ein früherer Ansatz[73] der mit WSDL vergleichbar ist, war Allaires WDDX (Web Distributed Data eXchange). Im Vergleich zu WSDL ist WDDX aber primitiver, da z.B. keinerlei Aussagen über Transportprotokolle getroffen werden. WDDX wird an dieser Stelle nicht näher vertieft. Weiterführende Informationen finden sich unter [All01].

[71] Network Accessible Service Specification Language
[72] Services Description Language/SOAP Contract Language
[73] vgl. (Java-Magazin, Ausgabe 5.2001, S.91f, Michael Knuth)

WSDL dient nach [Wey01a] zur Beschreibung von Kommunikationsendpunkten, Interfaces, Methodensignaturen, Schema für Datentypen und zur Flusskontrolle (siehe 4.5.5). Um dies zu verdeutlichen, wird das oben eingeführte Beispiel des MD5 Dienstes wieder aufgegriffen:

Die Methode `getMD5` hat in Java folgende Signatur:

```
public Sring getMD5(String toEncrypt)
```

Die Methode erwartet also eine Zeichenkette und gibt ebenfalls eine Zeichenkette zurück. Die entsprechende Beschreibung in WSDL sieht wie folgt aus:

```
<?xml version="1.0" ?>
    <definitions
        name="MD5Service"
        targetNamespace="http://localhost/MD5"
        xmlns:tns="http://localhost/MD5"
        xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema"
        xmlns="http://schemas.xmlsoap.org/wsdl/" >

<message name="IngetMD5">
    <part name="toEncrypt" type="xsd:string" />
</message>
<message name="OutgetMD5">
    <part name="return" type="xsd:string" />
</message>

<portType name="MD5ServicePortType">
    <operation name="getMD5">
        <input message="IngetMD5" />
        <output message="OutgetMD5" />
    </operation>
</portType>
<binding name="MD5Binding" type="MD5ServicePortType">
    <soap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="getMD5">
        <soap:operation soapAction="" />
        <input>
            <soap:body namespace="urn:MD5"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            use="encoded" />
        </input>
        <output>
            <soap:body namespace="urn:MD5"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            use="encoded" />
        </output>
    </operation>
</binding>
<service name="MD5">
    <documentation>Calculate MD5 Hash from any string</documentation>
    <port name="MD5ServicePort" binding="MD5Binding">
        <soap:address location="http://localhost:8080/soap/servlet/rpcrouter" />
    </port>
</service>
</definitions>
```

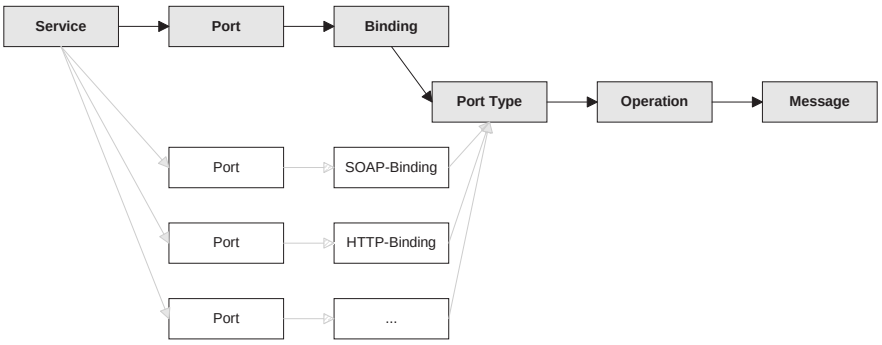

Abbildung 14: Beziehungen von WSDL Elementen (vgl. [Pü01])

Eine WSDL Datei besteht nach [CCMW01] aus folgenden Elementen:

- **Types**
  Definition der für den Nachrichtenaustausch benötigen Datentypen. Diesen Definitionen muss ein ”Typensystem” zugrunde liegen. Oft wird hier XML Schema verwendet.

  Das MD5 Beispiel verwendet ausschließlich den Datentyp String, der als einfacher Datentyp hier nicht definiert werden muss.

  Als Beispiel könnte für den Parameter, den die Methode `getMD5` erwartet, aber auch ein eigener Datentyp definiert werden.

```
<types>
    <schema targetNamespace="http://localhost:8080/MD5"
            xmlns="http://www.w3.org/2000/10/XMLSchema">

        <element name="MD5Request">
             <complexType>
                 <all>
                     <element name="toEncrypt" type="string"/>
                 </all>
             </complexType>
        </element>
     </schema>
</types>
```

- **Message**
  Eine abstakte und typisierte Definition der Nachricht, die verschickt bzw. empfangen werden soll. Im oben eingeführten Beispiel entspricht dem der folgende Ausschnitt:

```
<message name="IngetMD5">
    <part name="toEncrypt" type="xsd:string" />
</message>
<message name="OutgetMD5">
    <part name="return" type="xsd:string" />
</message>
```

  Eine Nachricht besteht aus einem oder mehreren logisch zusammenhängenden Teilen (parts). Diese Teile besitzen immer einen definierten Datentyp und repräsentieren im Kontext eines RPC-Aufrufs die Parameter bzw. den Rückgabewert der aufgerufenen Methode respektive Funktion. Die Nachrichtendefinitionen sollten möglichst abstrakt formuliert werden, was den Inhalt der Nachricht betrifft. Diese abstrakten Definitionen werden später durch das Binding auf ein konkretes Nachrichtenformat abgebildet. Durch die

zunächst abstrakte Formulierung, können dieselben Nachrichten auf mehrere konkrete Formate abgebildet werden. Eine solche Vorgehensweise ermöglicht eine bessere Pflegbarkeit, da Änderungen nur an einer Stelle vorgenommen werden müssen.

- **Port Type**
  Eine abstrakte Beschreibung der Operationen, die auf einem oder auf mehreren Kommunikationsendpunkten ausgeführt werden können.

```
<portType name="MD5ServicePortType">
    ...
</portType>
```

  WSDL unterstützt für den Nachrichtenaustausch vier Transportprimitive in Bezug auf den Kommunikationsendpunkt:

  - **One-way**
    Der Endpunkt erhält eine Nachricht (Messaging Fall).
  - **Request-response**
    Der Endpunkt erhält eine Nachricht und sendet daraufhin eine Antwortnachricht zurück (RPC Fall).
  - **Solicit-response**
    Der Endpunkt sendet eine Nachricht und erhält daraufhin eine Antwortnachricht zurück (wird nicht näher betrachtet).
  - **Notification**
    Der Endpunkt sendet eine Nachricht (wird nicht näher betrachtet).

- **Operation**
  Eine abstakte Beschreibung der Leistungen, die ein Service zur Verfügung stellt.

```
    <operation name="getMD5">
        <input message="IngetMD5" />
        <output message="OutgetMD5" />
    </operation>
```

- **Binding**
  Hier wird sowohl das konkrete Transportprotokoll wie auch das konkrete Nachrichtenformat für einen speziellen Port Type definiert und an diesen gebunden. Weiterhin werden abstrakte Operationsdefinitionen assoziiert.

```
<binding name="MD5Binding" type="MD5ServicePortType">
    <soap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="getMD5">
        <soap:operation soapAction="" />
        <input>
            <soap:body namespace="urn:MD5"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            use="encoded" />
        </input>
        <output>
            <soap:body namespace="urn:MD5"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            use="encoded" />
        </output>
    </operation>
</binding>
```

Das Binding muss an genau ein spezifisches Protokoll erfolgen und darf keine Informationen über die Adresse des Kommunikationsendpunktes beinhalten. WSDL unterstützt zur Zeit die Nachrichtenformate SOAP und MIME. Im Rahmen dieser Arbeit wird nur das SOAP Format näher betrachtet.

- **Service**
  Verbindet die verschiedenen Ports, die fachlich zusammengehören, aber nicht miteinander oder untereinander kommunizieren. Das heißt, dass das Ergebnis eines Ports nicht als Eingabe für einen anderen benutzt werden kann und vice versa. Es werden alle relevanten Kommunikationsendpunkte beschrieben.

```
<service name="MD5">
    <documentation>Calculate MD5 Hash from any string</documentation>
    ...
</service>
```

  Die Ports, die innerhalb eines Services definiert sind, haben folgende Eigenschaften: Hat ein Service mehrere Ports, die den gleichen PortType benutzen aber verschiedene Bindings oder Endpunkte besitzen, sind diese Ports Alternativen. Jeder Port stellt die gleiche semantische Funktionalität dar, die lediglich über ein anderes Transportprotokoll und/oder Nachrichtenformat genutzt werden kann. Somit kann derselbe Service für mehrere Bindings verfügbar sein. Dies erlaubt dem Nutzer eines Web Services die Art und Weise wie in Bezug auf Transportprotokoll und Nachrichtenformat frei zu wählen[74]. An dieser Stelle wird deutlich, welche Vorteile die abstrakte Definition sowohl der Nachrichteninhalte als auch der einzelnen Operationen besitzt.

- **Port**
  Beschreibt einen einzelnen Kommunikationsendpunkt durch die Angabe einer Bindingdefinition (Transportprotokoll und Nachrichtenformat) und einer Netzwerkadresse.

```
<port name="MD5ServicePort" binding="MD5Binding">
        <soap:address location=
                        "http://localhost:8080/soap/servlet/rpcrouter" />
    </port>
```

  Ein Port darf nicht mehr als einen Endpunkt spezifizieren. Darüber hinaus dürfen keine weiteren Binding Informationen angegeben werden.

Binding
Ist der Dienstnehmer im Besitz der WSDL-Datei (z.B. per E-mail, FTP oder UDDI), so ist er im Besitz aller notwendigen Informationen, um den angebotenen Dienst nutzen zu können. Er kennt die Methodennamen, deren Signaturen, die verwendeten Datentypen, die unterstützten Transportprotokolle und die Netzwerkadressen der Kommunikationsendpunkte. Hieraus lässt sich nun manuell oder automatisch ein lokales Proxy-Objekt (Stub) erzeugen, welches die gleichen Methoden aufweist wie das Originalobjekt. Die Methodenimplementationen enthalten, wie es beim klassischen RPC/RMI auch der Fall ist, die gesamte Funktionalität, um über das Netzwerk (hier unter Verwendung von SOAP) die korrespondierenden Methoden auf dem Server aufzurufen. Die praktische Umsetzung ist in Kapitel 5.5.3 detailliert erläutert. Dort finden sich auch Quellcodebeispiele für Implementationen in Java.

Zusammenfassung
WSDL beschreibt die Syntax, um entfernte Dienste dynamisch (siehe Dynamic Binding, Kapitel 4.3.3) nutzen zu können. Es ist im Hinblick auf die CORBA Welt mit der IDL (Interface Definition Language) vergleichbar. WSDL liefert jedoch keine semantischen Informationen. Dies bedeutet, dass zwar automatisch Code generiert werden kann, mit dem entfernte Methoden aufgerufen werden können, jedoch keine formale Beschreibung

[74] falls diese Kombination aus Transportprotokoll und Nachrichtenformat vom Service unterstützt wird

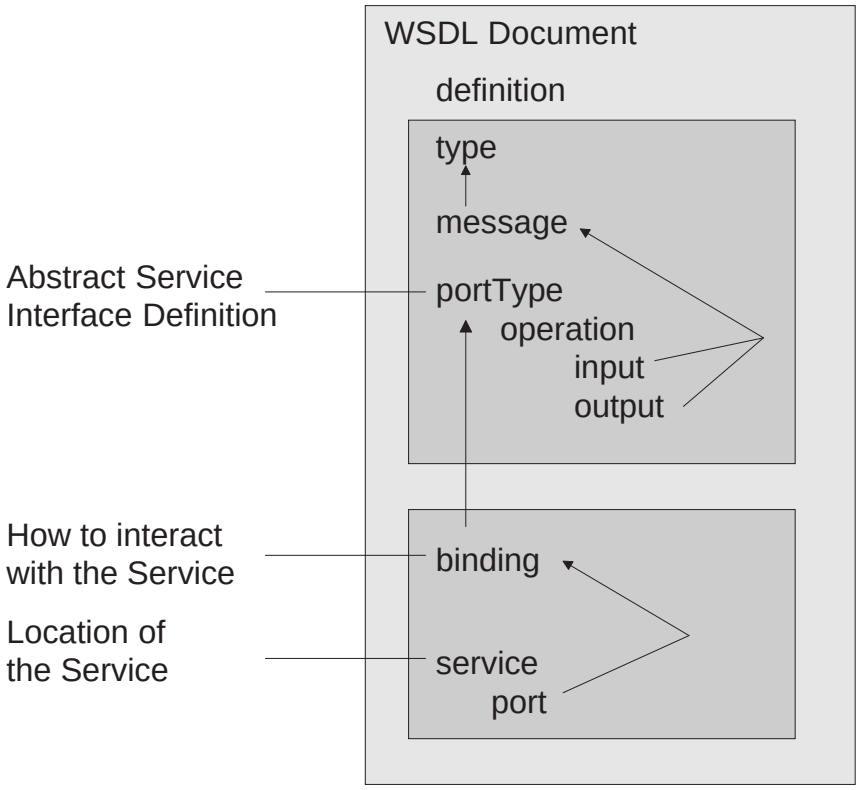

Abbildung 15: Struktur einer WSDL Datei (vgl. [Pü01])

dieser Methoden im Sinne ihrer Bedeutung (d.h. ihrer eigentlichen Aufgabe) vorliegt. Insofern ist immer noch der Mensch nötig, der aufgrund einer nicht formalen semantischen Beschreibung dieser Methode entscheidet, in welchem Kontext diese Methode aufgerufen werden muss. Dies macht ein volldynamisches Binden zur Zeit unmöglich.

4.4.5 UDDI als Verzeichnisdienst

”UDDI stands for Universal Description, Discovery and Integration, and is destined to become the standard way for businesses to share structured information about themselves and their services. [Ele01]

Das Akronym UDDI steht für Universal Description, Discovery and Integration. UDDI ist im Kontext von Web Services der globale Verzeichnisdienst (`global registry`), an welchem Dienste angemeldet werden können (`publish`) und in welchem nach Diensten gesucht werden kann (`find`). Dabei werden die Daten innerhalb des UDDI Verzeichnisses in einer ganz bestimmten Struktur abgelegt. Diese Struktur orientiert sich der Struktur von Unternehmen. Auch Unternehmen, die keine Web Services anbieten, können sich an einem UDDI Verzeichnis anmelden, da UDDI zunächst einmal universal ist. In diesem Kapitel wird UDDI ausschließlich in Verbindung mit Web Services betrachtet.

”The UDDI (Universal Description, Discovery and Integration) Project is a comprehensive, open industry initiative enabling businesses to (I) discover each other, and (II) define how they interact over the internet and share information in a global registry architecture. UDDI is the building block which will enable businesses to quickly, easily and dynamically find and transact with one another via their preferred applications.” [IBM01b]

Technisch gesehen ist UDDI eine verteilte Datenbank[75], die mittels SOAP Nachrichten manipuliert werden

[75] Es gibt mehrere UDDI Knoten, die ihre Daten untereinander abgleichen (”registered once, published everywhere” Prinzip).

kann. Die tatsächliche physische Datenhaltung kann mit Hilfe von relationalen Datenbanken oder sonstigen beliebigen Persistenzmechanismen realisiert werden. UDDI ist dabei nur die Schnittstelle, die die Struktur, welche nach außen hin sichtbar ist, vorschreibt und den Zugriff via SOAP ermöglicht. Es werden zunächst die Datenstrukturen vorgestellt. Danach wird auf die Operationen eingegangen, die auf einem UDDI Knoten ausgeführt werden können. Das Kapitel schließt mit einer kurzen Zusammenfassung in der auch die Vor- und Nachteile sowie eine kurze Prognose für die Zukunft enthalten ist. Die behandelte Version ist UDDI 1.0. Die Spezifikation 2.0 liegt zwar in einer endgültigen Version vor, es existieren aber bisher kaum Implementierungen. Auf die Unterschiede zwischen den beiden Versionen wird gegen Ende des Kapitels kurz eingegangen. UDDI wird von einem Konsortium entwickelt, dem zur Zeit über 300 Unternehmen angehören. Die offizielle Adresse ist http://www.uddi.org.

Datenstruktur
Dieser Abschnitt orientiert sich an [BHK+01] und [Wie01].

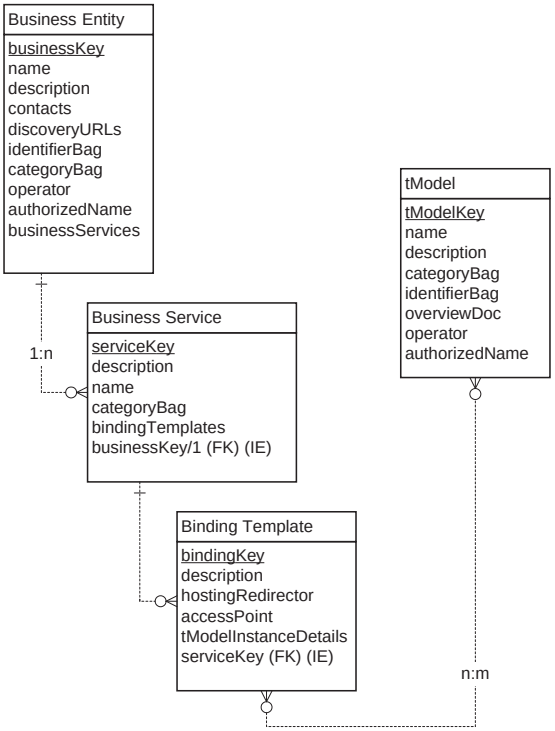

Abbildung 16: ER-Modell der UDDI Datenstruktur

Wie der Grafik zu entnehmen ist, gibt es vier Datenstrukturen die untereinander verknüpft sind. Alle aufgeführten Datenstrukturen besitzen eine XML-Repräsentation und sind mittels der XML Schema Definition festgelegt.

- **BusinessEntity (das Unternehmen)**
  Beschreibt das Unternehmen als solches. Hierzu gehört zwingend eine eindeutige Bezeichnung in Form eines Primärschlüssels (*Business-Key*), der Name der Unternehmung und der Name der Person, die für den UDDI Eintrag verantwortlich ist. Dann gibt es eine Reihe freiwilliger Informationen wie Kontaktangaben, eine textuelle Beschreibung des Unternehmens, Klassifizierungsbezeichnungen des Unternehmens und Verweise auf die *BusinessServices*, die das Unternehmen anbietet. Hier muss mindestens ein Eintrag stehen, da sonst die *BusinessEntity* nach einer bestimmten Zeit automatisch gelöscht wird.

- **BusinessService (Abteilungen, einzelne Dienstleistungen oder Produkte)**
  Ein *BusinessEntity* kann beliebig viele *BusinessServices* besitzen. Ein *BusinessService* beschreibt eine

”Serviceeinheit” einer Unternehmung. Dies kann eine Abteilung, ein Produkt oder eine Dienstleistung sein. Die genaue Ausprägung hängt von der Art des Unternehmens ab. Zwingende Angaben sind hier der *Business-Key* der referenzierten *BusinessEntity*, der *Service-Key* (Primärschlüssel), der Name und ein *BindingTemplate*. Zusätzlich können eine Beschreibung und eine Klassifizierung angegeben werden.

- **BindingTemplate (Zugriffsinformationen)**
  Hier wird die Kommunikationsschnittstelle für den betreffenden *BusinessService* beschrieben. Allerdings kann ein *BusinessService* beliebig viele *BindingTemplates* besitzen, wenn er mehrere Kommunikationsschnittstellen besitzt. Eine solche Schnittstelle könnte z.B. eine einfache Telefonnummer oder ein Dienstendpunkt[76] für einen Web Service sein. An dieser Stelle zeigt sich, wie generisch respektive universal UDDI ist. Die Mussfelder sind hier der *Binding-Key* (Primärschlüssel), der *Service-Key*, ein Verweis auf ein *tModel* und die Angabe eines Kommunikationsendpunktes oder eines anderen *BindingTemplates ((hostingRedirector)*. Optional kann eine Beschreibung des Templates angegeben werden.

- **tModel (Technische Informationen)**
  tModels sind relativ eigenständige, vom Unternehmen unabhängig definierte Service Schnittstellen. Zwischen *BindingTemplate* und *tModel* herrscht eine m:n Relation. *tModels* enthalten Metadaten, die den Web Service abstrakt beschreiben und ihn eindeutig mit einem *tModel-Key* bezeichnen. Somit bietet das *tModel* die Möglichkeit, nach kompatiblen Web Services zu suchen, die aufgrund ihres *tModel-Keys* zum eigenen System passen. Daher heißt der *tModel-Key* auch technischer Fingerabdruck (*technical fingerprint*) (vgl. [Rie01, S. 10 f.]). Beschrieben wird ein *tModel* durch den *tModel-Key* (Primärschlüssel), seinen Namen, den Operator (Betreiber der UDDI Registrierungsseite) und dem Namen der Person, die für den *tModel* Eintrag verantwortlich ist (*authorizedName*). Optional können eine Beschreibung, ein Übersichtsdokument, welches die Web Service Schnittstelle beschreibt, (*overviewDoc*) und Klassifizierungsinformationen angegeben werden. Im hier betrachteten Kontext von UDDI in Verbindung mit Web Services, verweist die Angabe im Feld *overviewDoc* auf die WSDL Datei, die den Web Service beschreibt.

  ”To invoke a Web service, you must know a service’s location and the kind of interface the service supports. The bindingTemplate indicates the specifications or interfaces a service supports through references to specification information. Such a reference is called a tModelKey, and the data structure encapsulating the specification information is called a tModel. tModels are reusable due to their use of references.” [BP01]

Um diese Datenstruktur etwas plakativer zu beschreiben, teilt man sie oft in die sogenannten ”white pages” (weiße Seiten), ”yellow pages” (gelbe Seiten) und ”green pages” (güne Seiten) ein. Die weißen Seiten sind ein einfaches Namensregister mit der Angabe von Kontaktinformationen (vergleichbar mit dem Telefonbuch). Die gelben Seiten sind (wie aus den ”gelben Seiten” der Deutschen Telekom bekannt) Branchenverzeichnisse, in denen nach verschiedenen Taxonomien gesucht werden kann. Die Einträge sind Referenzen auf die weißen Seiten. Die grünen Seiten bieten spezielle Informationen über das Geschäftsmodell und die Geschäftsprozesse des Unternehmens sowie technische Details zu den angebotenen Web Services. Auf die Datenstruktur von UDDI bezogen, stellt die Klasse *BusinessEntity* die weißen Seiten (”Wer bin ich”) und die Klasse *BusinessService* die gelben Seiten (”Was biete ich an”) dar. Das *BindingTemplate* und das *tModel* sind die grünen Seiten (”**So** kann man Geschäfte mit mir machen”).

Operationen
Die Operationen, die sich auf einem UDDI Knoten ausführen lassen, können grob in zwei Bereiche gegliedert werden: Veröffentlichen (*publish*) und Finden (*find*). UDDI definiert für diese Operationen ein API.

- **Publish**
  Ein Service Provider muss seinen Dienst und dessen Beschreibung veröffentlichen, damit er gefunden werden kann. Dies geschieht über ein API welches Methoden für das Publizieren von Diensten bereitstellt. Diese Kommunikation findet SOAP basiert und typischerweise über eine sichere Verbindung statt. Die vollständige API Spezifikation ist unter [UDD01] verfügbar.

[76] Typischerweise der Endpunkt der SOAP Nachrichten entgegennimmt.

Die wichtigsten API Funktionen sind die `save_*` Funktionen. Hiermit lassen sich die oben besprochenen Datenstrukturen zu einem UDDI Verzeichnis hinzufügen. Es gibt insgesamt vier dieser Funktionen: `save_business, save_service, save_binding und save_tModel`.

Als Beispiel sollen hier der oben eingeführte MD5 Dienst mit Hilfe der UDDI4J[77] API an einem UDDI Verzeichnis angemeldet werden. Hier werden nur die wichtigsten Code Ausschnitte gezeigt.

Quellcode 4.2: UDDI4J

```
1  UDDIProxy proxy = new UDDIProxy();
2
3  /*
4  UDDI Datenstruktur aufbauen
5  */
6
7  BusinessEntity be = new BusinessEntity("","MD5 Company");
8  be.setAuthorizedName("Hendrik Saly");
9  be.setDefaultDescriptionString("This is the MD5 Calculation Company");
10 Vector bes = new Vector();
11 bes.addElement(be);
12
13 try
14 {
15   String url = "https://ibmtestregistry/publishapi"
16   proxy.setPublishURL(url);
17   AuthToken at = proxy.get_authToken("user","password");
18
19   proxy.save_business(at.getAuthInfoString(), bes);
20 }
21 catch(Exception e)
22 {
23       System.out.println(e);
24 }
```

- **Find**
  Ein potentieller Dienstnehmer muss in der Lage sein, den Dienst, den er benötigt, aufgrund seiner Beschreibung und Spezifikation zu finden. Hier existiert ebenfalls ein API. Dieses API wird auch ”Inquiry[78] API” genannt. Die wichtigen Funktionen sind hier die `find_*` und die `get_*` Funktionen. Die `find` Funktionen suchen in der betreffenden Datenstruktur nach Einträgen, die mit den spezifizierten Suchparametern (z.B. Name) übereinstimmen. Es gibt vier Find Funktionen: `find_business, find_service, find_binding, find_tModel`.

  Die vier Get Funktionen `get_businessDetail, get_serviceDetail, get_bindingDetail und get_tModelDetail` liefern Detailinformationen zu den gefundenen Objekten. Die Kommunikation findet SOAP basiert aber typischerweise über eine nicht gesicherte Verbindung statt.

  Der Beispielcode zeigt, wie der zuvor registrierte MD5 Dienst gefunden werden kann. Zum Einsatz kommt hier die Microsoft UDDI API.

```
Dim reqMan As New UDDIEnv.RequestManager
Dim env As New UDDIEnv.Envelope
Dim respEnv As UDDIEnv.Envelope
Dim inqMsg As New UDDI10.find_business
Dim inqRsp As New UDDI10.businessList
Dim result As UDDI10.businessInfo

Set env.Plugin = inqMsg

reqMan.Mode = Custom
```

[77] http://oss.software.ibm.com/developerworks/projects/uddi4j
[78] eng. Abfrage, Nachfrage

```
reqMan.UDDI_Address = http://www-3.ibm.com/services/uddi/inquiryapi

' Suchtext festlegen
inqMsg.Name = "Suchtext"

' Anfrage abschicken
Set respEnv = reqMan.UDDIRequest(env)

' Antwort holen
Set respEnv.Plugin = inqRsp

' Ergebnisse ausgeben
For Each result In inqRsp.businessInfos
  Debug.Print result.Name
Next
```

Nachdem der Dienstnehmer einen passenden Dienst gefunden hat, kann er dessen Beschreibung (WSDL) nutzen, um sich an ihn zu binden. Wichtig ist hier, dass ab diesem Zeitpunkt das UDDI Verzeichnis nicht mehr Bestandteil des Kommunikationsmodells ist. Die Kommunikation zwischen Dienstanbieter und Dienstnehmer findet ausschließlich zwischen diesen beiden[79] statt. Es besteht eine Punkt-zu-Punkt (*Peer-to-Peer*) Verbindung. Der Dienstnehmer kann sich beliebig oft, solange die Dienstbeschreibung gültig ist, ohne zwingend notwendige Nutzung von einem UDDI Verzeichnis mit dem Dienstanbieter verbinden. Das Binding wird in den Kapiteln 4.4.4 und 5.5.3 näher beschrieben.

APIs, um auf ein UDDI Verzeichnis zugreifen zu können, gibt es u.a. von IBM, Microsoft und Bowstreet. Es existieren auch zahlreiche Open Source Implementierungen. Im Rahmen dieser Arbeit wird die IBM API (UDDI4J) benutzt, da sie in Java geschrieben ist und mit dem kostenlosen Web Service Toolkit von IBM ausgeliefert wird. Internetadressen zu diesen APIs, zu webbasierten UDDI Browsern, weiteren Toolkits und den zur Zeit existierenden öffentlichen UDDI Verzeichnissen sind unter http://www.soapclient.com zu finden.

Die Vorteile von UDDI sind vor allem die einheitliche Strukturierung der gespeicherten Daten und das verteilte Prinzip von UDDI (”registered once, published everywhere”). Dies bietet dem Dienstnehmer die Möglichkeit, sofort einen Ersatzservice zu finden, falls der ursprünglich gefundene und benutzte Dienst nicht mehr alle Kriterien erfüllt (z.B. Preis). Ferner ist natürlich auch UDDI durch die Verwendung von SOAP (also XML) von der Plattform und der Programmiersprache unabhängig. Als Nachteile können die bisher fehlende Standardisierung und eventuelle Datenschutz Probleme gesehen werden. Denn je mehr Daten über ein Unternehmen in der UDDI gespeichert sind desto effizienter funktioniert das ganze System. Unternehmen müssen sich gut überlegen, welche Unternehmensinformationen dort abgelegt werden und welche nicht. Es geht vor allem um Informationen zu Geschäftsprozessen, und zu internen Schnittstellen. Analysten rechnen mit einem Durchbruch von UDDI nicht vor 2003 (vgl. [Sch01]). Eine mögliche Alternative wäre ein LDAP[80] Verzeichnis in dessen Knoten die XML Daten gehalten werden. LDAP ist seit Jahren ein etablierter Standard. Das System ist stabil und weit verbreitet.

Versionsunterschiede
Die Version 2.0 bietet gegenüber der Version 1.0 folgende Neuerungen (vgl. [Cov01a]):

- **Präzisere Beschreibung der Unternehmensstruktur**
- **Bessere Unterstützung von Mehrsprachigkeit**
- **Detailliertere Kategorisierung von Unternehmenssparten**

[79] bzw. über SOAP Zwischenknoten
[80] Lightweight Directory Access Protocol

- **Mehr Suchfunktionalitäten**

Zusammenfassung
UDDI ist eine verteilte Datenbank, auf die mittels SOAP zugegriffen werden kann und die ein fest strukturiertes hierarchisches Datenmodell besitzt. Dieses ist hauptsächlich auf geschäftliche Nutzung ausgelegt. Was allerdings genau in den Datenstrukturen zu stehen hat, ist weitestgehend offen (universal) gehalten. Es lassen sich prinzipiell zwei Operationen auf ein UDDI Verzeichnis anwenden: Man kann Dienste veröffentlichen (publish) oder nach ihnen suchen (find).

4.5 Notwendige additive Mehrwertdienste

Einige wichtige Aspekte wurden bisher außer Acht gelassen, da sie innerhalb der Web Service Architektur nicht spezifiziert sind. So wurden bisher keinerlei Aussagen über die Sicherheit, die Dienstqualität, zu Transaktionen oder Kontextsensitivität (Sessions) und zur Verkettung einzelner Web Services gemacht. Alle diese Punkte sind aber sehr wichtig, falls sich Web Services als B2B Plattform durchsetzen sollen. Die im folgenden beschriebenen Mehrwertdienste sind bisher weder einheitlich spezifiziert noch liegen breite Erfahrungswerte vor. Daher ist das Kapitel eine momentane Bestandsaufnahme, die zum einen unterstreichen soll, welches Potential sich in den Web Services verbirgt, zum anderen auch zeigen soll, dass noch einige Dinge verbessert werden müssen. Gerade hier zeigt sich, neben den noch ausstehenden Standardisierungen von WSDL und UDDI, dass Web Services noch nicht bereit für den Einsatz ”im Großen” sind. Es sind aber trotz aller Kritik schon Lösungen in Sicht und vor dem Hintergrund der hohen Entwicklungsgeschwindigkeit ist es nur eine Frage der Zeit, bis endgültige Lösungen und Standards gefunden werden.

4.5.1 Sicherheit

Innerhalb des Web Service Stacks wird das Thema Sicherheit nicht spezifiziert. Die zentrale Komponente die von dieser Problematik betroffen ist, ist SOAP. Dort werden die eigentlichen Daten übertragen und wenn diese geschützt werden sollen, muss hier angesetzt werden. SOAP ignoriert das Thema Sicherheit ganz bewusst, da ein Hauptaspekt des Designs die Einfachheit des Protokolls ist. Wie man SOAP nun sicher macht, und dass es noch einen anderen Ansatzpunkt als SOAP selbst gibt, ist Thema dieses Kapitels.

Sicherheit meint in diesem Zusammenhang folgendes (vgl. [Jec01d]):

- **Vertraulichkeit (confidentiality)**
  Kein Dritter darf die Daten lesen können.
- **Berechtigung (authorization)**
  Der Nutzer muss die Berechtigung besitzen, den Dienst oder die Daten anfordern zu dürfen.
- **(Daten-)konsistenz (data integrity)**
  Es muss sichergestellt sein, dass die Daten nicht verändert wurden.
- **Glaubwürdigkeit des Ursprungs (message origin authentication)**
  Garantiert, dass die Daten vom Absender willentlich erstellt wurden.
- **Verbindlichkeit (non-repudiation)**
  Der Absender darf nicht abstreiten können, dass die Daten von ihm stammen.

Es ist aber nicht notwendig immer alle Kriterien gleichzeitig zu erfüllen, sondern es muss hier je nach Anwendungsfall entschieden werden, welcher Sicherheitsstandard für die jeweilige Nachricht erforderlich ist. Oftmals muss eine Nachricht gar nicht verschlüsselt, sondern nur signiert werden.

Um diese Kriterien technisch umsetzen zu können, existieren zwei verschiedene Ansätze, die Web Services sicher machen. Zum einen ist es möglich, diese Aufgabe an das darunterliegende Transportprotokoll zu delegieren, andererseits kann sie auch auf Applikationsebene gelöst werden. Dies ist in Abb. 17 dargestellt. Die erste Variante fußt auf der SSL (Secure Socket Layer) Technologie, die zweite auf XML Encryption und XML Signature.

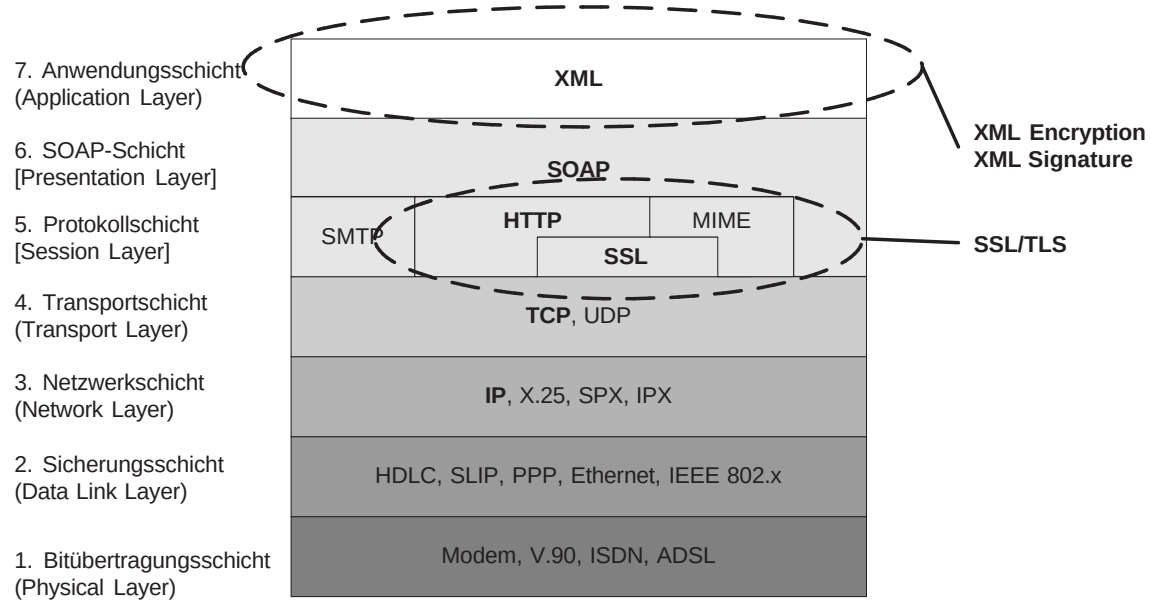

Abbildung 17: Einordnung von SSL und XML Encryption im Protokollstack (vgl. [Jec01d])

SSL

SSL steht für Secure Socket Layer[81] und ist ein von Netscape Communications Corp. entwickeltes Protokoll, um sichere Ende-zu-Ende Verbindungen im Internet zu ermöglichen. Es liegt zur Zeit in der Version 3.0 vor. Dabei ist zu beachten, das SSL 3.0 kein Standard, sondern nur ein IETF Working Draft ist. SSL hat sich aber als de-facto Standard etabliert und ist weit verbreitet im Einsatz. Es ist vom Anwendungsprotokoll unabhängig, was bedeutet, dass beliebige darüberliegende Protokolle über SSL sicher gemacht werden können. Es ist für die Darstellungs- und Anwendungsschicht transparent. Die bekannteste Kombination ist SSL und HTTP, die als HTTPS bezeichnet wird. Aber beispielsweise auch FTP, POP3 oder SMTP können über SSL übertragen werden. Genau diese Eigenschaft ist für Web Services respektive SOAP sehr wichtig, da SOAP vom Transportprotokoll unabhängig sein muss. Es kann also (fast) jedes Transportprotokoll, über welches SOAP Nachrichten verschickt werden, über SSL gesichert werden. In diesem Kapitel wird nur SSL in Verbindung mit HTTP betrachtet. Neben SSL existiert seit kürzerer Zeit noch TLS (Transport Layer Security), durch welches SSL abgelöst werden soll. TLS wird inoffiziell auch als SSL Version 3.1 bezeichnet und ist zu SSL 3.0 und 2.0 rückwärtskompatibel. SSL verschlüsselt aber nicht nur den Datenstrom, sondern beinhaltet zusätzlich eine zertifikatsbasierte Authentifizierung sowohl des Servers wie auch des Clients. Der Server muss sich gegenüber dem Client auf jeden Fall ausweisen, die Ausweisung des Clients gegenüber dem Server ist jedoch optional. Da SSL in der heute gebräuchlichen Verschlüsselstärke von 40 bit bzw. 56 bit relativ schwach ist, empfiehlt sich für sensitive Daten eine Kombination von SSL mit XML Encryption.

HTTPS

HTTPS ist somit kein neues Protokoll, sondern vielmehr eine Erweiterung von HTTP durch den Einsatz von Verschlüsselungstechnologie zwischen der Transport und der Anwendungsschicht [82]. Bisher wird HTTPS hauptsächlich im Web eingesetzt, um sensible Daten z.B. beim Online-Banking oder bei Bestellungen zu schützen.

[81] http://www.netscape.com/eng/ssl3/draft302.txt

[82] SSL arbeitet nach dem ISO/OSI Referenzmodell auf der Session Schicht (Session Layer)

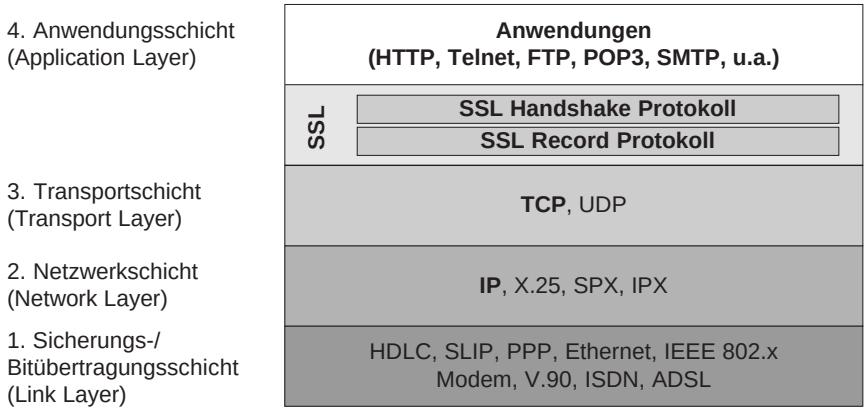

Abbildung 18: SSL im TCP/IP-Stack (vgl. [Jec01d])

Der Standardport[83] für HTTPS ist 443. Um eine sichere HTTPS Verbindung etablieren zu können, muss SSL sowohl vom Server, als auch auf dem Client unterstützt werden.

Die Abb. 19 zeigt, wie ein SSL Verbindungsaufbau abläuft.

Die Frage nach der Vertrauenswürdigkeit von Zertifikaten und deren Signierung durch eine Certification Authority (CA) wird hier nicht näher behandelt.

Der Vorteil von HTTPS ist die Transparenz gegenüber der Anwendung, wenn die Verbindung einmal besteht. Allerdings muss der Verbindungsaufbau auf der Clientseite durch die entsprechende Implementation unterstützt werden. Es ist aber kein Eingriff beim Verschicken einer SOAP-Nachricht erforderlich. Der große Nachteil ist der langsame Verbindungsaufbau, da hier jedes Mal der komplette Handshake abgehandelt werden muss. Außerdem bietet das SSL-Protokoll nur sichere Ende-zu-Ende Verbindungen, was unter dem Aspekt, dass auch SOAP Zwischenknoten (intermediaries) zugelassen sind, dann nicht mehr anwendbar ist. Werden SOAP Dokumente über verschiedenen Zwischenknoten geleitet, so muss die Verschlüsselung auf SOAP-Ebene stattfinden.

Die im Kapitel 4.4.1 besprochene HTTP Authentifizierung ist hier nicht notwendig, da der Client sich über ein SSL Zertifikat bereits dem Server gegenüber ausweisen kann. Generell ist der HTTP Authentifizierungsmechanismus für dienstorientierte Architekturen ohnehin nicht geeignet, da er darauf ausgelegt ist, Dateien und nicht Dienste zu schützen.

XML Encryption

Einen ganz anderen Ansatz als SSL verfolgt XML-Encryption. Da die gesamte Web Service Kommunikation respektive SOAP in XML stattfindet, kann das vom W3C entwickelte XML Encryption[84] benutzt werden. Hier wird das XML Dokument selbst (oder auch nur Teile davon) verschlüsselt. Die Sicherheit wird hier von der Anwendungsschicht übernommen. XML Encryption basiert darauf, dass ein Sender eine Nachricht verschlüsselt, die dann vom Empfänger wieder entschlüsselt werden muss. Dabei ist der Verschlüsselungsalgorithmus frei wählbar, so dass sowohl symmetrische wie auch asymmetrische Verfahren eingesetzt werden können. Diese Verschlüsselung garantiert die Vertraulichkeit der Daten, sagt aber zunächst nichts über die Identität der am Datenaustausch beteiligten Personen aus. Diese und weitere Aufgaben werden von XML Signature übernommen, welches weiter unten genauer behandelt wird. Der Schlüsselaustausch, die Verschlüsselungsalgorithmen und das Schlüsselmanagement sind nicht Bestandteil dieser Arbeit.

Um XML Encryption nutzen zu können, muss ein geeigneter Algorithmus ausgewählt werden. Die Ver-/Entschlüsselungsroutinen müssen in die jeweilige Anwendung integriert werden und die bereits oben angesproche-

[83] well known port
[84] W3C Working Draft

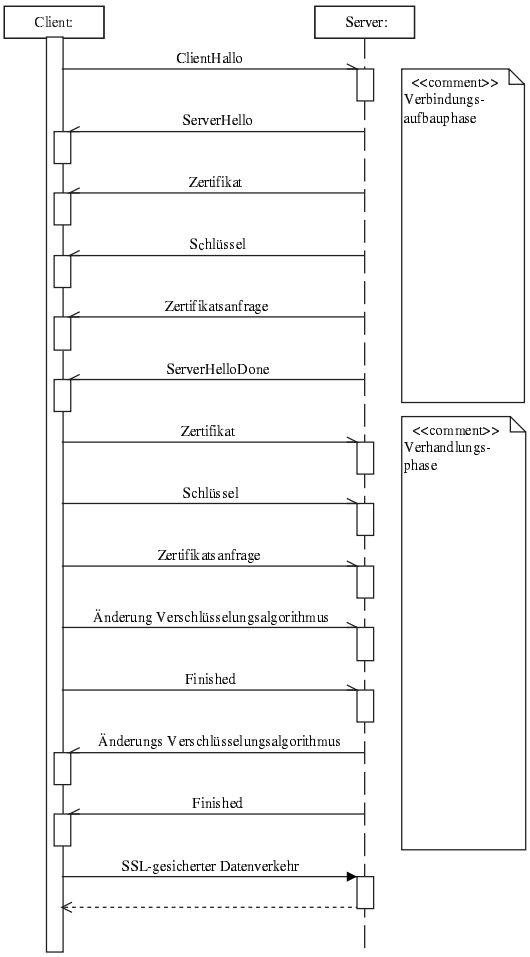

Abbildung 19: Aufbau einer SSL Verbindung

ne Verschlüsselungsgranularität muss festgelegt werden.

SOAP mit XML Encryption ist zunächst transparent, gegenüber der Anwendung, die SOAP Nachrichten verschicken möchte. Gleichwohl müssen aber die Ver- und Entschlüsselungsroutinen im Anwendungscode integriert sein. Dies muss aber nur einmal implementiert werden und kann dann anwendungsunabhängig benutzt werden.

Es ist zu beachten, dass XML Encryption eine strukturzerstörende Transformation ist, da die XML-Elemente (Tags), die die Struktur eines XML Dokumentes bilden, mitverschlüsselt und durch eine unstrukturierte Folge von Bytes ersetzt werden. Das heißt, dass eine Validierung gegen ein Schema sehr wahrscheinlich fehlschlagen wird. Gleichwohl ist eine verschlüsselte SOAP Nachricht (da nur der Body verschlüsselt wird) als Ganzes immer noch eine gültige SOAP Nachricht und kann gegen die Schemadefinition von SOAP erfolgreich validiert werden.

Beispielcode für XML Encryption ist unter [Jec01d] zu finden.

XML Signature

XML Signature kann Datenverfälschungen (*data integrity*) aufdecken, indem die Nachricht digital unterschrieben wird. XML Signature kann alleine oder in Kombination mit XML Encryption eingesetzt werden. Müssen Daten lediglich vor Manipulation geschützt und digital unterschrieben werden, so ist der Einsatz von XML Encryption nicht notwendig. XML Encryption und XML Signature können im Gegensatz zu SSL nicht nur für

sichere Ende-zu-Ende Verbindungen verwendet werden. Da sie direkt die SOAP Nachricht manipulieren, sind diese Verfahren auch dann geeignet, wenn die Nachricht über Zwischenknoten geleitet wird.

Das Prinzip von XML Signature ist folgendes: Zunächst muss das XML Dokument, welches zu signieren ist, kanonisiert werden. Kanonisierung bedeutet, dass das XML Dokument in eine bestimmte Form gebracht wird. Es werden alle irrelevanten Informationen entfernt, wie z.B. Leerzeichen, Zeilenumbrüche, Tabulatoren usw. Dies ist deshalb wichtig, da laut SOAP Spezifikation jeder Zwischenknoten genau diese Modifikationen vornehmen darf. Deshalb muss vor der Signierung das Dokument in eine 'minimale' Form gebracht werden. Signiert wird die kanonische Repräsentation, aber übertragen wird das Originaldokument. Über die kanonische Repräsentation der zu signierende Nachricht wird dann ein sogenannter Digest (z.B. SHA1 und MD5) gebildet. Wird diese Nachricht auch nur geringfügig geändert, so verändert sich dieser Digest relativ stark. Mathematisch gesehen sind die Digestalgorithmen Einweg-Hashfunktionen, die eine beliebig lange Nachricht in einer Art Kurzform repräsentieren. Dieser Digest hat normalerweise eine von der Ursprungsnachricht unabhängige konstante Länge. Dann wird dieser Digest mit dem privaten Schlüssel des Absenders verschlüsselt und wird so entsteht die digitalen Unterschrift.

Der digitalen Unterschrift liegt ein Public Key Verfahren zugrunde. Der Absender verschlüsselt den Digest mit seinem privaten Schlüssel. Der Empfänger kann diesen verschlüsselten Digest mit dem öffentlichen Schlüssel des Absenders entschlüsseln. Er berechnet über die eigentliche Nachricht zunächst deren kanonische Repräsentation und danach den Digest (die Digestalgorithmen müssen hierbei die gleichen sein). Dann vergleicht er diesen mit dem soeben entschlüsselten Digest. Stimmen sie überein, so kann sich der Empfänger sicher sein, dass die Nachricht während der Übertragung nicht verändert wurde. Weiterhin ist sicher, dass die Nachricht von der Person stammt, die den privaten Schlüssel besitzt, der das Gegenstück des öffentlichen Schlüssels ist, den der Empfänger verwendet hat. Jetzt muss nur noch gewährleistet sein, dass der private Schlüssel auch der Person gehört, die vorgegeben hat, der Absender der Nachricht gewesen zu sein. Dies wird durch Zertifikate garantiert, die von einer CA vergeben werden. Die Person muss sich dort persönlich ausweisen, um ein solches Zertifikat zu erhalten. Auf diesen Vorgang wird hier nicht näher eingegangen.

Die digitale Unterschrift muss nach [Jec01d] folgende Eigenschaften besitzen:

- **Fälschungssicherheit: Unterschrift kann nicht durch Dritte erzeugt werden**
  Der private Schlüssel einer Person ist ausschließlich dieser Person bekannt. Daher kann von Personen, die nicht im Besitz des privaten Schlüssels sind, die Unterschrift nicht gefälscht werden.
- **Glaubwürdigkeit: Willentliche Unterschrift**
  Da der Absender die Nachricht in einem extra Arbeitsschritt unterschreibt und hierzu meist auch ein Passwort eingegeben werden muss (der private Schlüssel ist durch ein Passwort gesichert), findet die Unterschrift bewusst und damit willentlich statt.
- **Unveränderbarkeit: Unterschrift und Dokument bilden Einheit**
  Da der Hashcode des Dokuments unterschrieben wird und dieser eine Repräsentation des Dokuments ist, kann dies als Einheit bezeichnet werden.
- **Transienz: Unterschrift ist nicht wiederverwendbar**
  Da die Unterschrift an das Dokument gebunden ist, kann sie nicht für andere Dokumente benutzt, also wiederverwendet werden.
- **Dauerhaftigkeit: Unterschrift kann nicht zurückgezogen werden**
  Da die Unterschrift beim Empfänger zwar digital aber real und persistent vorliegt, kann der Unterzeichner die Unterschrift nicht leugnen.

Beispielcode für XML Signature ist unter [Jec01d] zu finden.

Authentifizierung
Insbesondere kommerzielle bzw. sicherheitssensitive Dienste müssen einem Zugangsschutz unterliegen, der sicherstellt, dass nur berechtige Nutzer den Dienst in Anspruch nehmen können. Somit wird ein feingranulares

	Vertraulichkeit	Berechtigung	Datenkonsistenz	Glaubwürdigkeit des Ursprungs	Verbindlichkeit
SSL	ja	ja	ja	ja	ja
XML - Encryption	ja	nein	nein	nein	nein
XML - Signature	nein	ja	ja	ja	ja

Tabelle 2: Vergleich von Verschlüsselungstechnologien

Authentifizierungssystem notwendig. Wie oben bereits erläutert, ist der HTTP Authentifizierungsmechanismus nur bedingt einsetzbar, da er relativ primitiv und nur auf den Schutz von Dokumenten ausgelegt ist. Ferner sollen Web Services respektive SOAP vom Transportprotokoll unabhängig sein und der HTTP Authentifizierungsmechanismus steht innerhalb von anderen Transportprotokollen nicht zur Verfügung. Es muss also auf der SOAP Ebene oder höher eine Authentifizierungsmechanismus integriert werden. Da die SOAP Spezifikation diesen Punkt aber nicht berücksichtigt, bleibt nur der Weg, dieses Problem durch einen proprietären SOAP Header oder über XML Signature zu lösen. Ein Authentifizierungsdienst wäre eine weitere Lösung. Dieser würde dann auf der Anwendungsebene stattfinden und eine Art Session Handling erfordern (vgl. Kapitel 4.5.4).

Weiterführende Informationen hierzu und zu anderen sicherheitsrelevanten Themen bezüglich SOAP, bietet IBM mit seinen ”SOAP Security Extensions” unter http://www.trl.ibm.com/projects/xml/soap/.

4.5.2 Quality of Service

Unter ”Quality of Service” werden alle Arten von Dienstqualitäten subsumiert, die für einen bestimmten Dienst notwendig sind. Bekannt ist der Begriff aus der Netzwerktechnik, um z.B. garantierte Bandbreiten oder garantierte Maximalverzögerungszeiten zu beschreiben. Im Kontext der Web Services sind vor allen Dingen zwei Merkmale wichtig:

- **Nachrichten müssen garantiert und nur einmal ankommen (reliable one-time message delivery)**
  Zuverlässiger Nachrichtenaustausch ist für geschäftskritische Anwendungen unverzichtbar. Zuverlässig heißt auf der einen Seite, dass die Nachrichten tatsächlich ankommen und nicht verloren gehen und dass sie vor allen Dingen genau einmal ankommen. Typisches Szenario ist ein Bestellvorgang im Internet. Der Nutzer schickt die Bestellung ab und erhält zunächst keine Antwort. Er schickt die gleiche Bestellung noch einmal ab. Möglicherweise war das Netzwerk überlastet und die Bestätigung der ersten Bestellung hat ihn nicht sofort erreicht. Ein unzuverlässiger Datenaustausch hätte in diesem Beispiel eine zweifach ausgeführte Bestellung zur Folge. Außerdem sind weitere Fehlerszenarien denkbar, in denen z.B. ganze Anfragen oder Antworten einfach ”auf dem Weg” verloren gehen. Dies alles ist natürlich innerhalb von B2B Anwendungen nicht erwünscht. Da die Standard Internet Protokolle aber unzuverlässig sind und Web Services über diese abgewickelt werden, stellt sich die Frage, wie hier ein zuverlässiger Datenaustausch realisiert werden kann. Hierfür gibt es zwei Ansätze: Einerseits können die Transportprotokolle zuverlässig gemacht bzw. nur zuverlässige genutzt werden. Andererseits kann Zuverlässigkeit auch innerhalb der Anwendung umgesetzt werden.

  Den ersten Ansatz verfolgend entwickelt IBM zur Zeit das HTTPR (reliable HTTP) Protokoll. Es ist ein auf HTTP basierendes und zu HTTP kompatibles zuverlässiges Protokoll. Nähere Informationen findet man unter [TPC01]. Ob dieser Weg der richtige ist und sich HTTPR durchsetzen wird, bleibt abzuwarten. Bisher existieren noch keine stabilen Implementationen.

  Der Ansatz, die Komplexität, die nötig ist, um Zuverlässigkeit zu garantieren, in die Anwendung zu verlagern, macht zwar vom Transportprotokoll unabhängig, aber die Entwicklung der Anwendung erheblich aufwändiger. Die SOAP Extensions bieten Unterstützung für Reliable Messaging. Dort kann so etwas über den SOAP Header realisiert werden. Sonst gibt weder einen Standard, noch finden sich Implementierungsvorschläge oder Beispiele.

- **Verfügbarkeit der Dienste**
  Die Verfügbarkeit von Web Services unterscheidet sich prinzipiell nicht von der Problematik der Verfügbarkeit im Web. Man muss auch hier die Serversysteme redundant auslegen, die Hardware richtig dimensionieren, ein Loadbalancing durchführen und Fail-over Szenarien definieren. Der Unterschied liegt darin, dass Web Services kritische Systeme darstellen. Sie sind unter Umständen Bestandteil des Geschäftsprozesses oder übernehmen sonstige geschäftskritische Aufgaben. Nicht nur die generelle Verfügbarkeit spielt eine Rolle, sondern auch Faktoren wie Antwortzeiten, Wiederanlaufzeiten nach einem Ausfall, Backupstrategien und nicht zuletzt die Stabilität.

Wie zu sehen ist, befindet sich die Web Service Technologie in Bezug auf ”Quality of Service” noch in einer frühen Entstehungsphase. IBM nimmt sich dessen immerhin mit dem Vorschlag von HTTPR an, aber weitere Aktivitäten sind zur Zeit nicht erkennbar.

4.5.3 Transaktionen

Eine Transaktion ist ein Satz von Anweisungen, die entweder alle oder gar nicht ausgeführt werden. Schlägt eine Anweisung fehl, so muss das System in den Zustand gebracht werden, der vor Beginn der ersten Anweisung geherrscht hat.

Bekannt sind Transaktionen vor allen Dingen aus dem Datenbankbereich und den Großrechnersystemen. Ein typisches Beispiel für die Notwendigkeit von Transaktionen in Geschäftsprozessen ist z.B. eine Überweisung. Wenn Geld von einem Konto auf ein anderes überwiesen werden soll, so definiert man die Subtraktion des Betrages vom Ursprungskonto und die Addition des Betrages auf das Zielkonto als Transaktionen. Entweder es werden beide Vorgänge ausgeführt oder keiner. Es wäre denkbar, dass das Geld zwar vom einen Konto abgebucht, aber aufgrund eines Fehlers auf dem Zielkonto nicht ankommt. In diesem Falle wäre das Geld verloren und das System würde sich in einem inkonsistenten Zustand befinden, hätten wir die gesamte Aktion nicht als Transaktion definiert.

Verallgemeinert ausgedrückt, müssen Transaktionen dem ACID Konzept genügen (vgl. [Ley01a]):

- **Atomarität** (**A**tomiticy)
  Der als Transaktion definierte Satz von Anweisungen ist atomar (nicht teilbar). Es werden entweder alle Anweisungen oder keine ausgeführt.
- **Konsistenz** (**C**onsistency)
  Eine erfolgreiche Transaktion erhält die Datenbankkonsistenz. Sie versetzt eine Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand. Falls durch eine Transaktion Integritätsbedingungen verletzt werden, wird die Transaktion abgebrochen und die Datenbank im Ursprungszustand belassen.
- **Isolation** (**I**solation)
  Parallel ablaufende Transaktionen sehen nichts von dieser Parallelität. Jede Transaktion sieht während ihrer Ausführung stets nur einen konsistenten, während der Transaktion von außen unveränderlichen Datenbankzustand. Sie läuft also isoliert von anderen Transaktionen ab.
- **Dauerhaftigkeit/Persistenz** (**D**urability)
  Sobald eine Transaktion erfolgreich beendet wurde, ist der von ihr bewirkte Zustandsübergang dauerhaft. Er darf auch nicht durch einen Systemausfall nicht verloren gehen.

Im Normalfall laufen Transaktionen in einer geschlossenen Umgebung (z.B. Datenbank oder Host) ab und sind eher von kurzer Dauer. Das Transaktionsmodell und die ACID Eigenschaften auf das Internet zu transferieren ist schwierig. Die Infrastruktur ist hier viel offener und unzuverlässiger als innerhalb der geschlossenen Umgebungen. Es gibt praktisch keine Möglichkeit vorauszusagen, wie lange eine Nachricht im Netz unterwegs sein wird

und ob sie überhaupt ankommt. Transaktionen über das Internet zu realisieren stellt neue Anforderungen, die man sowohl bei den eigentlichen Transaktionsvorgängen als auch beim Design der Anwendung beachten muss. Ein erster Punkt bezieht sich auf die Tatsache, dass Transaktionen im Internet verteilte und eher langlaufende Transaktionen sind. Das Anwendungsdesign ist in der Hinsicht betroffen, dass bestimmte Abstriche in Bezug auf das ACID Konzept gemacht werden müssen.

Transaktionen im Bereich Web Services sind wie oben bereits erwähnt verteilte Transaktionen. Für verteilte Transaktionen ist der Begriff ”two phase commit” (zwei Phasen Commit, 2PC) von entscheidender Bedeutung. Das zugehörige Zwei-Phasen-Commit-Protokoll hat die Aufgabe, die Atomarität und Dauerhaftigkeit der ACID-Eigenschaften von verteilten Transaktionen, die auf mehreren Knoten ausgeführt werden, zu garantieren (vgl [SS00, S. 270 ff.]).

Die hier als Knoten bezeichneten Systeme müssen nicht notwendigerweise Datenbanken sein, sondern es sind beliebige Ressourcen (z.B. Dateisysteme) denkbar. Diese transaktionale Ressourcenverwaltung wird von Transaktionsmonitoren[85] gesteuert.

Nach [Du01] gibt es zahlreiche Implementationen für das 2PC Protokoll. Unter anderem sind das LU6.2, XA, OTS und JTA/JTS. Keine dieser Implementationen lässt sich direkt auf die Infrastruktur des Internets anwenden, da dass 2PC Protokoll einerseits auf einer komplexen Programm-zu-Programm Kommunikation beruht und zum anderen weil eine bestimmte darunterliegende Infrastruktur nötig ist, die im Internet nicht existiert.

Um die aufgezeigten Probleme zumindest teilweise zu lösen, existieren mehrerer Ansätze (vgl. [Du01]):

- **Transaction Internet Protocol (TIP)**
  TIP wurde entworfen, um verteilte Transaktionen über das Internet zu ermöglichen. Es liegt zur Zeit in der Version 3.0 vor und ist durch den RFC 2371 spezifiziert. Es folgt dem sogenannten ”two-pipe” Modell, welchem die Trennung von Protokoll und Interapplikationskommunikation zugrunde liegt. Die TIP Architektur basiert auf dem 2PC Konzept und beschreibt einen übergeordneten Transaktionsmonitor (TM), der lokale, untergeordnete TM koordiniert. Die lokalen TM können, unabhängig vom Kommunikationsprotokoll der betreffenden Applikation, an einer Transaktion teilnehmen. TIP ist hauptsächlich dafür ausgelegt, dass der TM eines Knotens mit einem TM eines anderen Knotens über das Internet kommuniziert. Bisher ist unklar, welche Hersteller TIP unterstützen werden und ob es sich durchsetzen kann.

- **Transaction Authority Markup Language (XAML)**
  XAML ist ein herstellerunabhängiger, offener Standard, der die Koordination von Business Transaktionen über das Internet ermöglicht, vor allen Dingen im Hinblick auf Web Services. Der Standard definiert eine Reihe von XML Nachrichten und Interaktionsmodellen, die den Web Services transaktionale Eigenschaften ermöglichen sollen.

  ”XAML is designed for the coordination of transactional web services, not XML transportation and packaging issues. XAML will work with standard XML-based service transport protocols, including W3C XML Protocol (XP), SOAP and ebXML transport protocol.” [XAM01]

  An der Standardisierung sind u.a. HP, IBM, Oracle und SUN beteiligt. Gerüchten zufolge hat die XAML Arbeitsgruppe allerdings ihre Arbeit (vorübergehend) eingestellt. Es ist zur Zeit unklar, wie die Zukunft von XAML aussehen wird.

- **Business Transaction Protocol (BTP)**
  Das BTP wird zur Zeit von der OASIS[86] spezifiziert. Detaillierte Informationen finden sich unter [OAS02]. Das BTP lockert das den Transaktionen zugrunde liegende ACID Konzept. Es lässt u.a. zu, das kleine lokale Teile der Transaktion dem 2PC Protokoll folgen, die dann zu größeren Transaktionen zusammengefügt werden, die aber dann nicht ACID konform sind. Ferner kann eine Transaktion mehrere erfolgreiche Ausgänge haben und sie muss nun nicht mehr notwendigerweise dauerhaft oder isoliert sein (Abschwächung des ACID Konzeptes). Um die Unzuverlässigkeit der Internetkommunikation zu berücksichtigen, ist es zugelassen dass, Transaktionsteilnehmer zeitweise unerreichbar sind. Das BTP basiert auf XML. In

[85] z.B. Tuxedo, IMS oder CICS
[86] Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/

der BTP Terminologie ist ein Atom die Bezeichnung für eine atomare Transaktion. Für die Atome gelten die Regeln des 2PC. Die Atome können zu Cohesions (Kohäsion) aggregiert werden, die nicht mehr atomar sein müssen (vgl. [Kü01]).
”Cohesions erlauben die ’Aufweichung’ der ACID-Anforderungen” [Kü01]. Der Ausgang einer Cohesion wird durch die Business-Regeln und nicht durch die starre Infrastruktur bestimmt. Nach [Kü01] ist ”BTP **die** Lösung für transaktionale Web Services”.

Bisher konnte sich noch kein Vorschlag durchsetzen. Dies bedeutet, dass es hier keinen Standard gibt und sich demzufolge alle großen Hersteller momentan noch zurückhalten. TIP liegt schon seit Juli 1998 als RFC vor und konnte sich bisher nicht etablieren. Vor dem Hintergrund einer möglichen Einstellung von XAML scheint BTP die favorisierte Variante zu sein. Tyler Jewell von BEA Systems sagt dazu: ”Currently, there aren’t any infrastructures that support BTP, but it should rapidly be adopted in the upcoming year.” [Jew01]

4.5.4 Kontextsensitivität

Kontextsensitivität meint zum einen die technische Umsetzung, um Dienste zustandsbehaftet zu machen, und andererseits die Fähigkeit der Dienste einen bestimmten Kontext, in dem sie aufgerufen werden zu erkennen und dynamisch darauf zu reagieren. Technisch gesehen ist Kontextsensitivität das Sammeln von externen Daten (z.B. über den Nutzer) und deren Verfügbarkeit während der gesamten Interaktionsphase. Um letzteren Punkt umzusetzen, muss eine Art Kontextpropagierung oder Sessionhandling eingeführt und umgesetzt werden.

Kontextsensitivität
Die Sun ONE Architektur mit ihren ”Smarten Web Services” unterscheidet sich in genau diesem Punkt von den ”normalen” Web Services. Sie definieren schon von vornherein eine Kontextsensitivität. Dies sind Informationen darüber ”welcher Anwender mit welcher Rolle, mit welcher Lokation welchem Profil und anderen Kriterien den Service aufgerufen hat ...” [Sar01, S. 42]. Sun ONE wird im Kapitel 3.4.2 detailliert behandelt.

Die folgenden Punkte sind mögliche Kriterien, die für einen kontextsensitiven smarten Web Service nützlich sein können (vgl. [Sar01, S. 40 ff.]):

- Identität des Nutzers
- Die Rechte des Nutzers
- Präferenzen, die für den Dienst definiert wurden
- Sicherheits- und Geschäftsaspekte des Nutzers
- Lokation des Nutzers (physischer Standort)
- Informationen über das Endgerät des Nutzers
- bestimmte Vereinbarungen zwischen Service Provider und Nutzer

Prinzipiell läßt sich diese Kontextsensitivität mit der Personalisierung von Webangeboten vergleichen. Hier müssen ebenfalls Daten über den Nutzer bzw. seine Interessensgebieten vorliegen, um ein personalisiertes Angebot bereitstellen zu können.

Verwendet man nicht die Sun ONE Architektur, muss die Kontextsensitivität je nach Bedarf und Ausprägung des Dienstes selbst definiert und implementiert werden. Hier spielt die im nächsten Abschnitt beschriebene Kontextpropagierung eine wesentliche Rolle.

Kontextpropagierung
Da SOAP, wenn es über Standard Internetprotokolle versendet wird, meist zustandslos ist, muss der Kontext über Zusatzinformationen, von Nachricht zu Nachricht weitergegeben werden (vgl. HTTP Sessions).

Man unterscheidet dabei fünf Arten von Kontexten (vgl. [Cem01]):

- **Application Context**
  In diesem Kontext befinden sich Daten, die für eine Applikation global sind. Die Lebensdauer entspricht der des Servers. Diese Informationen müssen nicht propagiert werden
- **Call Context**
  Daten, die für einen Aufruf eines Web Service gelten. Die Lebensdauer beträgt daher nur einen Aufruf.
- **User Context**
  Dieser Kontext umfasst alle Daten, die sich auf einen Nutzer beziehen (Stammdaten). Die Lebensdauer ist prinzipiell nicht beschränkt. Der Nutzer muss nicht notwendigerweise ein Mensch, sondern kann auch eine Maschine respektive ein anderes Programm sein.
- **Application Session Context**
  Hier sind Beziehungen zwischen User- und Callkontext gemeint. Ein Nutzer kann wiederum mehrer Calls (Sessions) gleichzeitig geöffnet haben. Der Application Session Kontext hält Informationen wie z.B. welche Nachricht für welchen Benutzer wurde versandt und in welchem aktuellen Zustand sich der Nutzer befindet. Die Charakteristik dieses Kontextes hängt stark von der Semantik der Anwendung ab.
- **Transaction Context**
  Dieser Kontext gehört in den Bereich der Transaktionen und wird typischerweise in der Application Session mitverwaltet. Daher ist eine explizite Propagierung nicht notwendig.

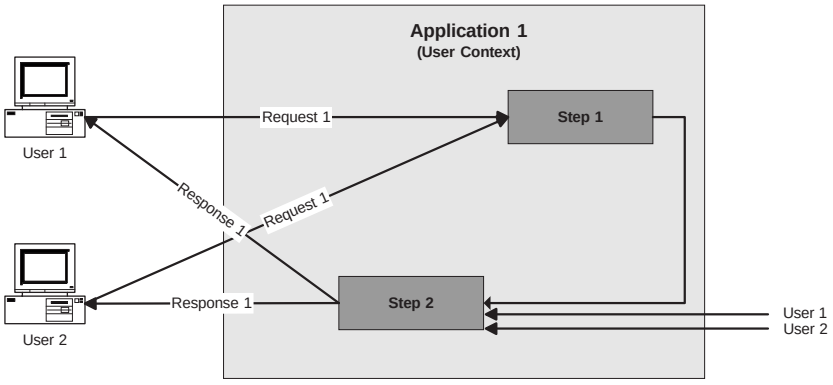

Abbildung 20: Usercontext (vgl. [Cem01])

Welche Möglichkeiten gibt es nun, den Call-/User-/und Application Session Kontext zu propagieren? Die Lösung ist ein Session Identifier (Session-ID). Dieser ist aus dem Umfeld der Web Programmierung bekannt und wird dort über HTTP von Seite zu Seite weitergegeben. Im Falle der Web Services muss die Session-ID von Nachricht zu Nachricht weitergeben werden und eigentlich vom Transportprotokoll unabhängig sein. Eigentlich deshalb, weil HTTP hier trotzdem ein Faktor ist. Es gibt somit zwei Möglichkeiten, eine Session-ID zu propagieren:

- **HTTP**
  Wie bereits oben diskutiert, gibt es mit HTTP und Session Variablen schon langjährige Erfahrung im Web Bereich. Dazu zählt die Variablenübergabe via GET/POST und vor allem mit Hilfe der Cookies. Diese Verfahren lassen sich analog auf SOAP Nachrichten anwenden, solange sie über HTTP übertragen werden. Es ist also nicht vom Transportprotokoll unabhängig, dafür aber einfach und erprobt. HTTP Cookies vom MS SOAP Toolkit noch nicht unterstützt.

- **über SOAP selbst**
  Es gibt auch die Möglichkeit die Session-ID innerhalb der SOAP Nachricht zu übertragen. Dies hat den Vorteil, dass es wirklich unabhängig vom Transportprotokoll ist. Es muss allerdings entschieden werden, ob die Session-ID als Meta Information in den SOAP Header oder als Teil der Nutzinformation in den SOAP Body geschrieben wird. Letzteres hat den Nachteil, dass jede Methode/Funktion den Identifier als Parameter akzeptieren muss. Dies ist sehr aufwändig und nicht generisch, aber die einzige Möglichkeit den Identifier in den SOAP Body zu schreiben. Der Vorteil ist die Unabhängigkeit von den ”Header Processoren” und den verschiedenen SOAP Implementierungen. Diese haben nämlich Interoperabilitätsprobleme, wenn es um die Auswertungen von SOAP Headern geht. Dort kann der Session Identifier nämlich ebenfalls platziert werden. Ferner müssen spezielle Header Processoren sowohl auf Client- wie auf Serverseite angestoßen werden, um den Header zu analysieren. Diese Methode hat den Vorteil, dass sie für die Anwendung transparent ist und dass die Session-ID unabhängig vom eigentlich Inhalt (SOAP Body) der Nachricht, propagiert werden kann.

  Die AXIS SOAP Implementierung[87] von Apache unterstützt sowohl die automatische Verarbeitung von Cookies als auch die Propagierung mittels des SOAP Headers (über sogenannte Handler).

4.5.5 Verkettung von Web Services

Web Services können zu einem neuen Web Service kombiniert werden. Diese relativ einfache Tatsache hat IBM dazu veranlasst, die Web Services Flow Language (WSFL) zu entwickeln. Wie sich schon am Namen erkennen läßt, handelt es sich nicht nur um eine Sprache, die eine einfache Aggregation von Web Services ermöglicht, sondern darüber hinaus können komplexe Prozesse (*Flows*) definiert werden. Dies würde z.B. ein Outsourcing von Geschäftsprozessen ermöglichen. WSFL liegt zur Zeit in der Version 1.0 vor und wird hauptsächlich von Frank Leymann entwickelt. Mit XLANG[88] versucht auch Microsoft eine Sprache zu schaffen, die ähnliches leisten kann. Zum Zeitpunkt des Schreibens liegen keine Informationen über einen praktischen Einsatz der beiden Konzepte vor. Weder WSFL noch XLANG sind bisher standardisiert oder haben breitere Verwendung gefunden. In Rahmen dieser Arbeit wird nur auf WSFL eingegangen.

WSFL
Eine WSFL Dokument ist ein XML Dokument, welches die Verkettung von Web Services als ein Flussmodell (*Flow Model*) oder ein Globales Modell (*Global Model*) definiert. Beide Modelle haben eine nach außen hin öffentliche Schnittstelle und eine interne Struktur, die die Komposition des nach außen hin sichtbaren Dienstes aus anderen Diensten repräsentiert (vgl. [Ley01b]).

Die beiden Modelle unterscheiden sich folgendermaßen:

”The first type (flow models) specifies the appropriate usage pattern of a collection of Web Services, in such a way that the resulting composition describes how to achieve a particular business goal; typically, the result is a description of a business process. The second type (global models) specifies the interaction pattern of a collection of Web Services; in this case,the result is a description of the overall partner interactions.” [Cov01b]

”[...]WSFL supports two types of composition and choreography: Flow models: describes business processes; Global models: describe overall partner interactions. Flow models - Describe how to choreograph the functionality provided by a collection of Web services to achieve a particular business need Global models - Describe how a set of Web services interact with each other.” [Cov01b].

Die Flussmodelle beschreiben einen Geschäftsprozess, wohingegen die Globalen Modelle die Wechselwirkung zwischen den zu kombinierenden Web Services beschreiben.

Die Sprache definiert Konstrukte einer Programmiersprache. Unter anderem gibt es in WSFL eine Auswahl (*Transition Conditions*), Wiederholung (*Loops*) und Funktionen für Nebenläufigkeit (*Forks and Parallelism*).

[87] http://xml.apache.org/axis/
[88] http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

Ferner sind rekursive Strukturen möglich.

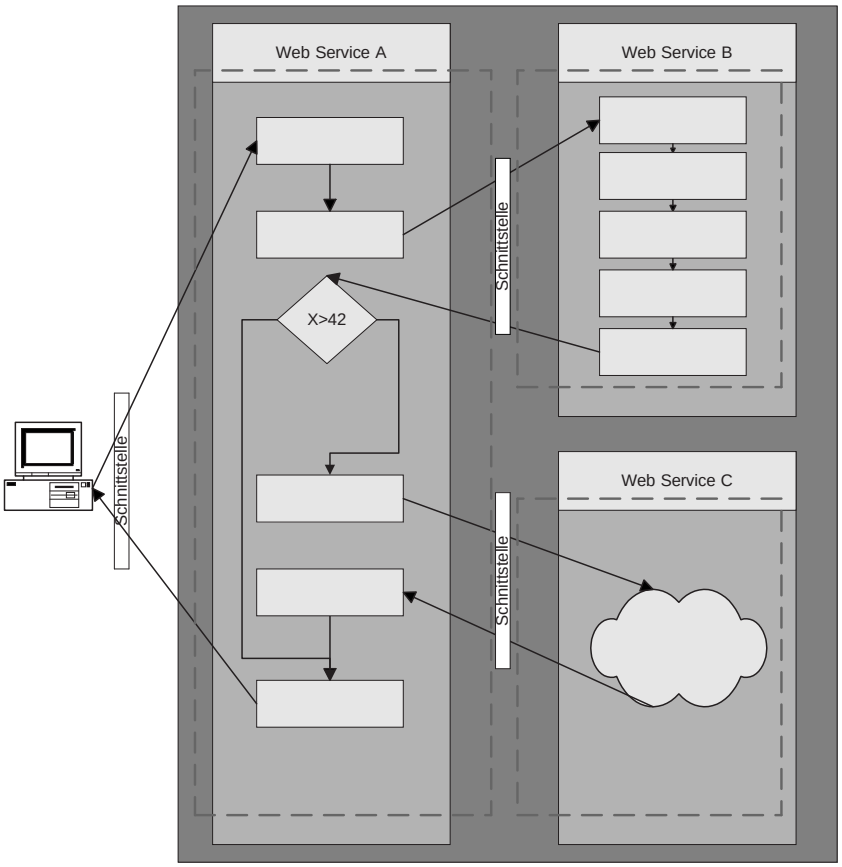

Abbildung 21: Aggregation von Web Services

Außer einem White Paper von IBM (vgl. [Ley01b]) gib es praktisch keine weiteren Informationsquellen zum Thema WSFL. Ferner fehlen Tools um WSFL Dateien zu erstellen und zu interpretieren. Erfahrungsberichte über eine prototypische oder produktive Verwendung von WSFL liegen zur Zeit ebenfalls nicht vor. WSFL ist die jüngste aller in dieser Arbeit vorgestellten Technologien, der aber der Ansicht des Autors nach ein wichtiger Stellenwert innerhalb der Web Service Architektur zuteil werden wird. Dies ist zum einen dadurch begründet, dass es eine Sprache zur Definition von Web Service Aggregation geben muss und zum anderen wird durch eine solche Technologie erst eine tatsächliche Prozessabbildung auf die Infrastruktur des Internets möglich. Um die hier beschriebenen Aufgaben erfüllen zu können, muss WSFL standardisiert werden. Gartner erwartet in naher Zukunft eine Vereinigung von WSFL und XLANG. Das Resultat soll dann an ein Standardisierungsgremium übergeben werden.

”Gartner expects IBM and Microsoft to jointly agree to submit a proposal to W3C that combines XLANG and WSFL by year-end 2001” [Smi01]

5 Praktische Umsetzung des Web Service Konzepts

In den letzten Kapiteln wurde das Konzept der Web Services theoretisch besprochen. In diesem Kapitel findet eine praktische Umsetzung dieses Konzepts statt. Eine bereits bestehende Java-basierte Anwendung wird um eine Web Service Schnittstelle erweitert und lässt sich so von einer Microsoft-basierten Anwendung heraus ansprechen. Das Kapitel stellt einen Leitfaden dar, mit dessen Hilfe die Umsetzung und Implementierung Schritt für Schritt nachvollzogen werden kann. Es kann damit als Vorlage für die eigene Web Service Entwicklung dienen.

Zunächst wird die Zielsetzung erläutert, die im Rahmen dieses Kapitels erfüllt werden soll. Darauf folgt die konkrete Aufgabenstellung und eine kurze Beschreibung der exemplarischen Java Applikation ”Java Insurance Company (JIC)”. Im Anschluss wird erläutert, wie die JIC um eine Web Service Schnittstelle erweitert wird. Es werden Voraussetzungen definiert, Entwicklungswerkzeuge vorgestellt und die nötigen Beschreibungsdateien erstellt. Eine Veröffentlichung des Dienstes in einem UDDI Verzeichnis vervollständigt den Web Service Entwicklungsprozess. Dies alles findet im Rahmen der ”serverseitigen Implementierung” statt. Auf der Clientseite werden ebenfalls die Voraussetzungen definiert, die nötig sind, um den Web Service nutzen zu können. Anschließend wird auf die verwendete SOAP Implementierung von Microsoft und auf die exemplarische Clientanwendung eingegangen. Eine Diskussion über die gesammelten Erfahrungen schließt das Kapitel ab.

5.1 Zielsetzung

Die prototypische Implementierung soll zunächst zeigen, dass das Konzept praktisch umsetzbar ist und für reale Einsatzzwecke angewendet werden kann. Der Schwerpunkt liegt darauf, die Integrationsfähigkeit zu demonstrieren. Diese wird durch Interoperabilität und dem einfachen Anbinden einer bestehenden Anwendung erreicht. Die Interoperabilität von Web Services wird hier durch den Einsatz unterschiedlicher Programmiersprachen und Betriebssysteme gezeigt. Die Anwendungsintegration wird dadurch demonstriert, dass eine bereits bestehende Anwendung (JIC) um eine Web Service Schnittstelle erweitert wird. Es wird dargelegt, was hierfür nötig ist, wie aufwändig es ist und wo Probleme liegen. Weiterhin soll herausgestellt werden, dass die Kommunikation ausschließlich über die (bereits vorhandene) Infrastruktur des Internets stattfindet. Es ist nicht nötig, besondere Voraussetzungen zu schaffen oder die Infrastruktur um besondere Komponenten zu erweitern. Durch diese Punkte grenzen sich Web Services u.a. von CORBA ab.

5.2 Aufgabenstellung

Die ”Java Insurance Company” ist eine auf dem JWAM Framework (vgl. Kapitel 5.3) aufbauende Demonstrationsanwendung, die Versicherungsprämien für Automobile berechnen und Versicherungsanträge verwalten kann. Diese Funktionalität ist bisher nur über einen Desktop-Client (Abb. 23) und über eine Weboberfläche verfügbar gewesen. Die Versicherungsfirma möchte nun ihre Dienste auch als Web Service anbieten, so dass Anwendungen anderer Unternehmen direkt oder über ein UDDI Verzeichnis auf diese Funktionalitäten zugreifen können. Ein Kunde der Versicherungsfirma ist die WST Bank. Sie bietet Leasing von Automobilen und versichert diese bei der Java Insurance Company. Seitens der WST Bank muss die Clientfunktionalität des Web Services mit Visual Basic realisiert und in ein Excel-Sheet eingebunden werden. Für die Sachbearbeiter der WST Bank läuft das Ausrechnen der Prämie und das Abschließen der Versicherung transparent ab. Es findet also primär eine Kommunikation zwischen Anwendungen statt. Über diesen Weg soll die Funktionalität bereitgestellt werden, die auch die Web Schnittstelle der Anwendung bietet. Die Desktop Variante bietet weitergehende Möglichkeiten, die jedoch nur für den Sachbearbeiter der JIC relevant sind und daher über die Web Service Schnittstelle nicht zur Verfügung stehen. Dazu zählt u.a. die Verwaltung der Versicherungsanträge.

Untenstehende Grafik stellt eine Übersicht über das Gesamtsystem dar:

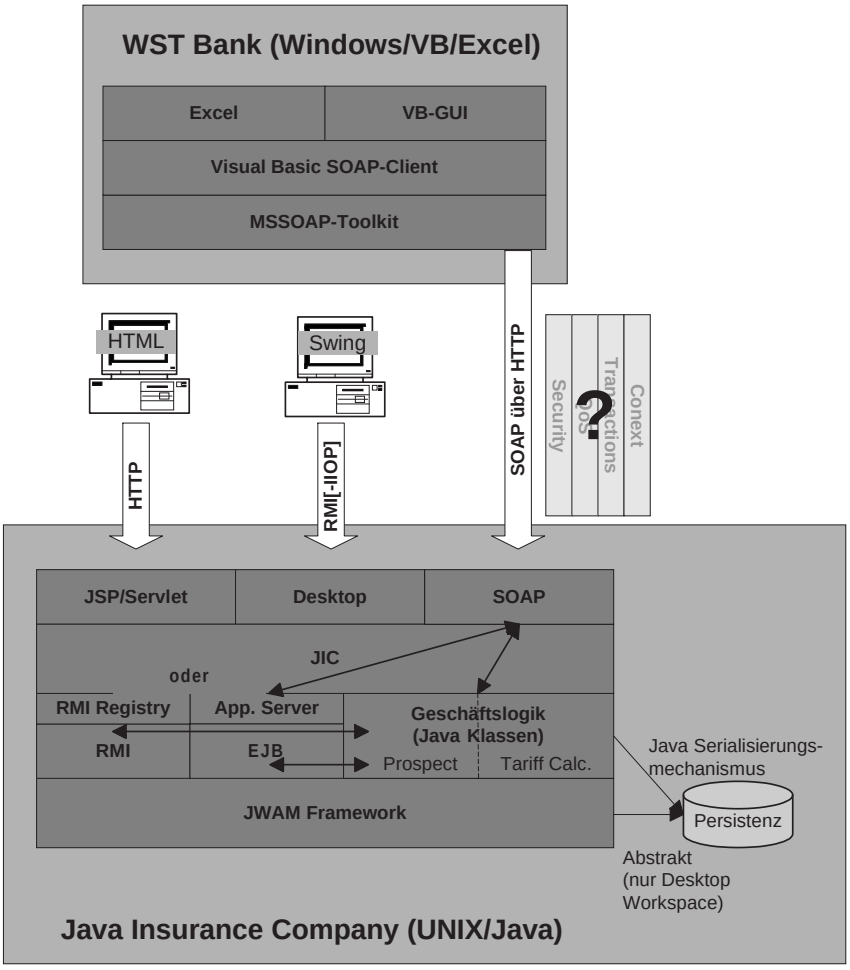

Abbildung 22: Übersicht über die prototypische Umsetzung

5.3 Java Insurance Company

”Das Tarifrechner-Beispiel demonstriert, wie mit dem JWAM-Rahmenwerk Multi-Frontend-Systeme ohne Duplizierung fachlicher Funktionalität entwickelt werden können. Als fachlicher Anwendungsbereich wurde die Tarifberechnung für eine KFZ-Versicherung gewählt. Über das Web-Frontend hat ein Interessent die Möglichkeit, sich verschiedene Tarife für sein Fahrzeug durchrechnen zu lassen. Findet er das Angebot attraktiv, so kann er sein Interesse bekunden. Nach Eingabe seiner Kontaktdaten (Anschrift, Telefon, E-Mail), kontaktiert ihn das Call-Center der Versicherung. Die Mitarbeiter des Call-Centers verwenden ein grafisches Applikations-Frontend und kein Web-Frontend. Das Beispiel ist nach WAM (Werkzeug, Automat, Material) entwickelt worden.” [RW02]

Die Java Insurance Company ist eine Anwendung, die auf dem JWAM Framework der APCON WPS[89] basiert. Das JWAM Framework unterstützt die objektorientierte Softwareentwicklung nach dem Werkzeug-Automat-Material (WAM) Ansatz (vgl. [Zü98]). Dieser verfolgt eine Metapher, in der Werkzeuge auf Materialien arbeiten und Automaten diesen Prozess automatisieren können. Werkzeuge bestehen dabei aus einer Interaktionskomponente, die im direkten Kontakt mit dem Material steht und eine Visualisierungskomponente, die das Aussehen des Werkzeugs bestimmt. Materialien sind Arbeitsgegenstände und oftmals Objekte aus der realen Welt, die sich mit Werkzeugen bearbeiten lassen. In dem hier verwendeten Beispiel sind die Materialien die zu versichernden Fahrzeuge und die Versicherungsanträge.

[89] Die APCON WPS gehört zur itelligence AG. Weitere Informationen finden sich unter http://www.jwam.de.

JWAM ist ferner ein Multichanneling-Framework. Das bedeutet, dass auf die (einmal vorhandene) Geschäftslogik über unterschiedliche Kanäle zugegriffen werden kann. Zur Zeit existieren die Kanäle Desktop (SWING) und Web-Oberfläche (über JSP/Servlets). Allerdings ist die Multichanneling-Fähigkeit nicht vollständig generisch, so dass für jede konkrete Anwendung, die auf dem Framework aufbaut, zusätzlicher Code geschrieben werden muss. In diesem Kapitel wird dem Framework nun in vergleichbarer Weise ein weiterer Kanal hinzugefügt. Der Kanal ermöglicht die Kommunikation über SOAP und erlaubt somit einen Zugriff auf die Funktionalität des Frameworks von anderen Anwendungen aus. Dieser Kanal wird allerdings nicht generisch implementiert, da dies aufgrund der Architektur des Frameworks nicht möglich ist. Die Abb. 25 gibt einen Überblick über die Klassen der Java Insurance Company.

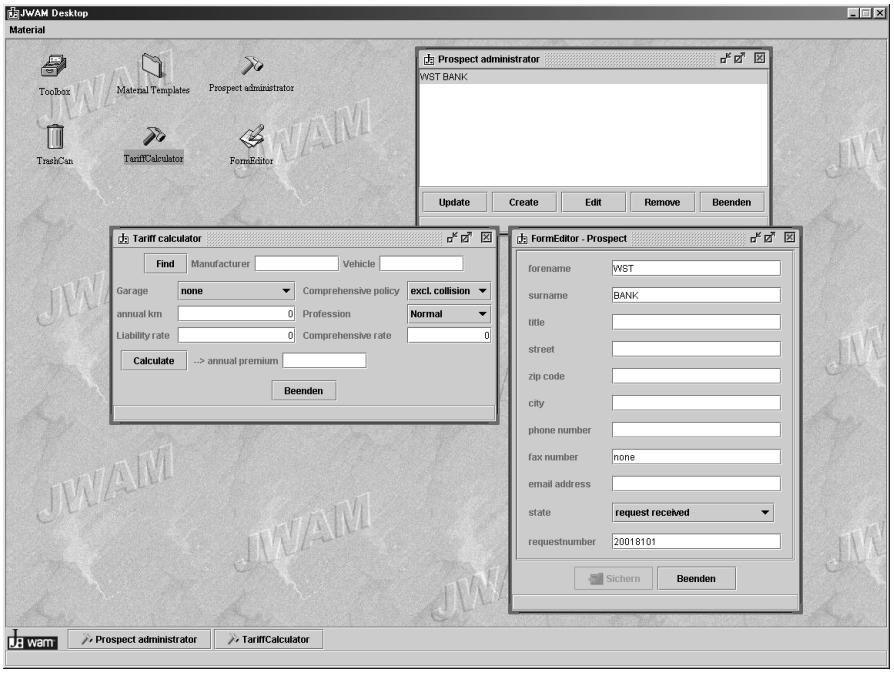

Abbildung 23: JIC SWING/Desktop Client

Im folgenden wird die Funktionalität und die Architektur der JIC besprochen. Die JIC besteht aus zwei funktional unterschiedlichen Komponenten:

- **Der Tarif Rechner (Tariff Calculator)**
  Er führt die Berechnung der Versicherungsprämie durch. Die erforderlichen Daten sind u.a. der Fahrzeugtyp, die Fahrleistung, und die Art der Unterstellung.
- **Die Vertragsverwaltung (Prospect Administration)**
  Hier werden die Anfragen für Versicherungen und Verträge verwaltet. Diese Daten sind im Gegensatz zur Tarif Berechnung zustandsbehaftet. Die Persistenz der Daten wird über den Java Serialisierungsmechanismus realisiert.

5.4 Serverseitige Implementierung

Dieses Kapitel beschreibt die Schritte, die auf der Serverseite durchgeführt werden müssen, um der JIC eine Web Service Schnittstelle hinzuzufügen. Es wird zunächst auf die Voraussetzungen eingegangen, die nötig sind, um eine Anwendung als Web Service verfügbar zu machen. Danach wird das Web Service Toolkit von IBM kurz vorgestellt. Darauffolgend werden die Schnittstellen der Anwendung besprochen, über die auf ihre Funktionalität zugegriffen werden kann. Die SOAP Implementierung muss konfiguriert und an die Schnittstellen der

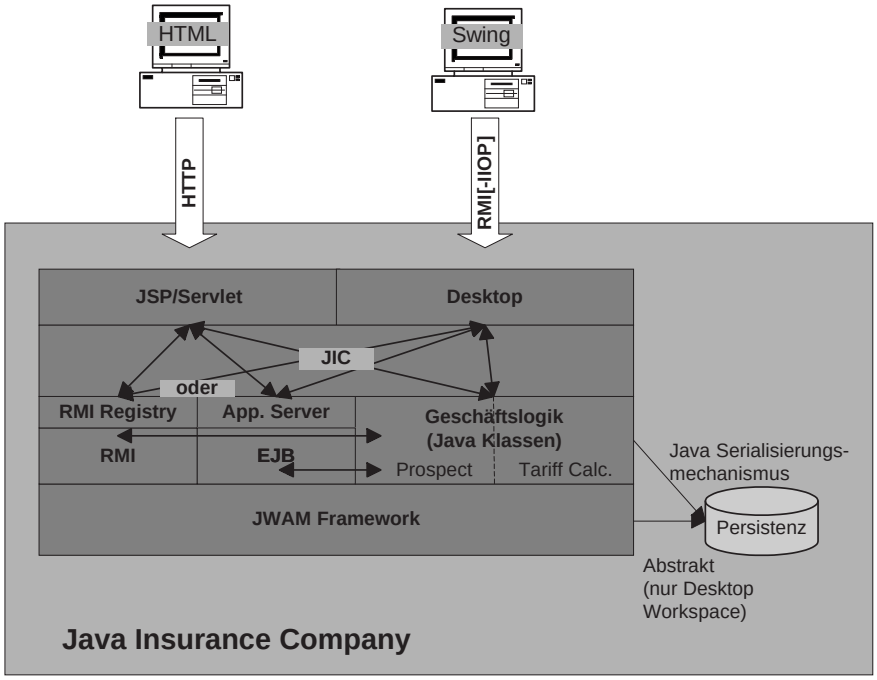

Abbildung 24: Bisherige JIC Architektur

Anwendung angepasst werden. Nach diesen Schritten ist der Dienst bereits nutzbar, aber noch nicht selbstbeschreibend. Daher wird an diesem Punkt die WSDL Beschreibung generiert. Um den Dienst finden zu können, wird am Ende des Kapitels der Dienst in einem UDDI Verzeichnis veröffentlicht.

5.4.1 Voraussetzungen

Die serverseitigen Voraussetzungen für den JIC Web Service sind:

- **JWAM Framework und JIC Anwendung V1.6**
  Hierzu gehören alle Klassen und Bibliotheken und die entsprechende Laufzeitumgebung (JDK 1.3, jBoss Application Server).
- **Apache SOAP Implentierung V2.2**
  Da die Anwendung in Java geschrieben ist, wird hier die ebenfalls in Java geschriebene Apache SOAP Implementierung (vgl. Kapitel 4.4.3) verwendet.
- **Tomcat V3.2**
  Servlet und JSP Container, der für die Apache SOAP Implementierung benötigt wird, die diese Servlets verwendet. Außerdem wird der im Tomcat enthaltene Web Server für die HTTP Kommunikation verwendet.
- **Zusätzliche Bibliotheken**
  - XML Parser Xerces V1.43
  - Java Mail
  - JavaBeans Activation Framework
  - Java Secure Socket Extensions (JSSE)

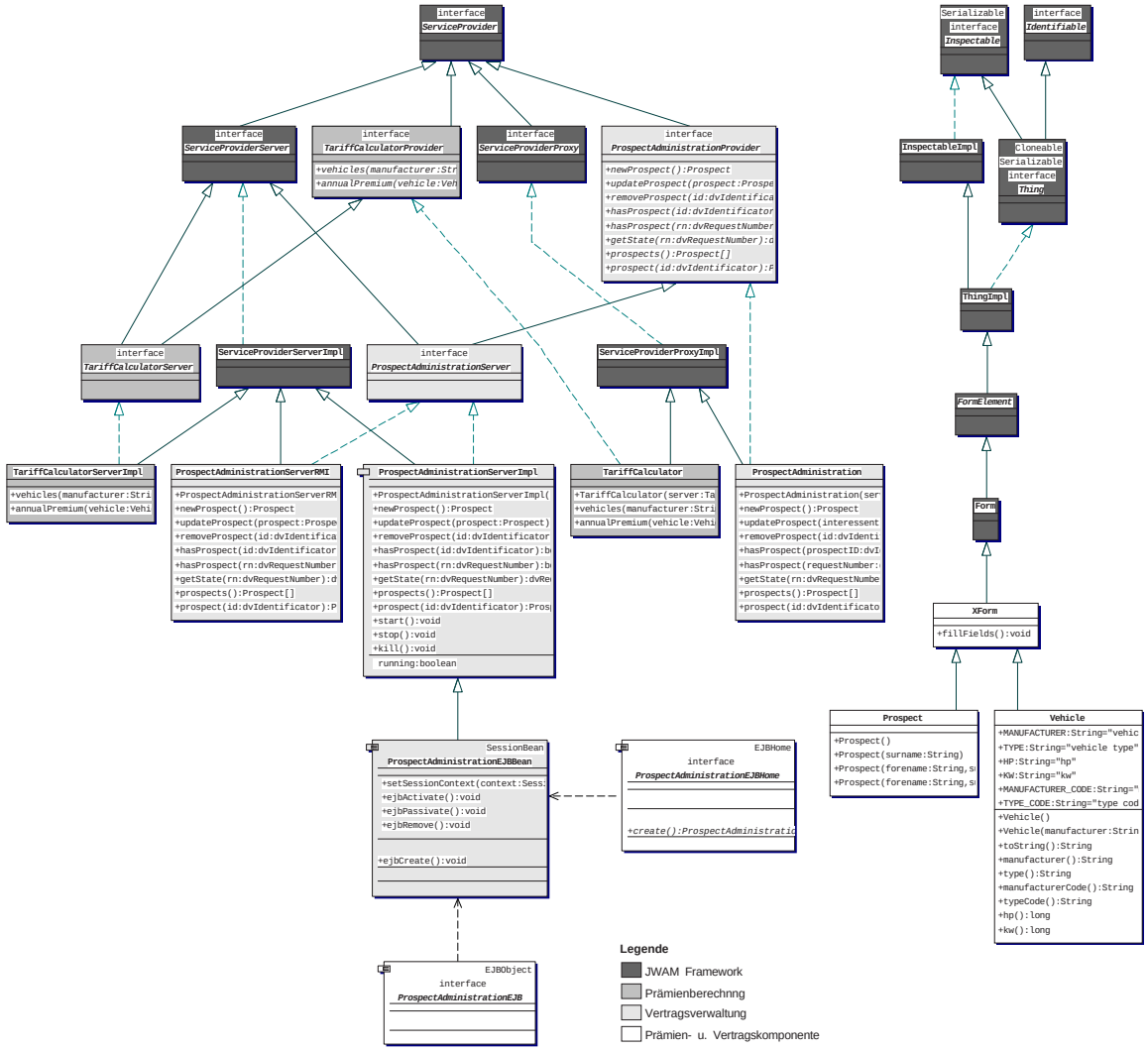

Abbildung 25: JIC Klassendiagramm

5.4.2 IBM Web Service Toolkit

Das frei erhältliche Toolkit[90] beinhaltet alles, was für die server- oder clientseitige Entwicklung von Web Services auf der Basis von Java notwendig ist. Es gehören sowohl IBM-eigene wie auch externe Bibliotheken und Programme zum Lieferumfang. Erstgenannte sind vor allen Dingen die WSDL, UDDI, MQSeries und HTTPR APIs, der WebSphere Application Server sowie die Beispielanwendung ”Gourmet2Go” und Tools zum automatischen Erzeugen von WSDL und Stub Dateien. An externen Bibliotheken sind u.a. Apache SOAP und AXIS enthalten. Im Rahmen dieser Arbeit werden nur einzelne Teile des Toolkits verwendet. Diese werden an den entsprechenden Stellen im weiteren Verlauf des Kapitels näher erläutert.

5.4.3 Schnittstelle zur Anwendung

Der hier verfolgte Ansatz ist der in Kapitel 4.3.1 beschriebene Bottom-Up Ansatz. Die Geschäftslogik besteht bereits in Form der JIC Anwendung und muss nun nach außen hin über eine SOAP Schnittstelle verfügbar

[90] http://www.alphaworks.ibm.com/tech/webservicestoolkit/

gemacht werden. Um diese äußere Schnittstelle schaffen zu können, muss aber zuerst eine oder mehrere innere Schnittstellen der Anwendung gefunden werden, die die gewünschte Funktionalität bereitstellen. Ist die Gesamtfunktionalität, die der Web Service nach außen besitzen soll, auf mehrere interne Schnittstellen verteilt, so muss eine ”Fassade” (Facade Pattern, vgl. [Ga94, S. 185 ff.]) erstellt werden, die diese Teilfunktionalitäten zusammenführt.

Aus obiger Aufgabenstellung ergeben sich für die Web Service Schnittstelle folgende Anforderungen:

1. Fahrzeugabfrage
2. Prämienberechnung für ein Fahrzeug
3. Versicherungsvertrag anfordern
4. Statusabfrage für angeforderte Versicherungen

Die Punkte 1 und 2 werden von der Tarifrechnerkomponente und die Punkte 3 und 4 von der Vertragsverwaltungskomponente übernommen.

Aus dem Klassendiagramm der Java Insurance Company gehen für die beiden Komponenten folgende Schnittstellen hervor:

1. **Schnittstelle Tarifrechnerkomponente**

   - Klasse: de.jwamexample.tariffcalculator.service.TariffCalculator
   - Interface: de.jwamexample.tariffcalculator.service.TariffCalculatorProvider

   Die Funktionalität wird durch die Methoden `annualPremium()` und `vehicles()` zur Verfügung gestellt. Sie besitzen folgende Signatur:

```
public float annualPremium (Vehicle vehicle, long annualDistance,
                            int garageType, boolean isComprehensive,
                            int liabilityRate, int comprehensiveRate,
                            dvProfession profession)


public Vehicle[] vehicles (String manufacturer, String type, int hp, int kw)
```

   Innerhalb der Signaturen finden sich u.a. die Datentypen `Vehicle` und `dvProfession`. Das sind anwendungsspezifische Datentypen. Ein `Vehicle` repräsentiert ein Fahrzeug und die Klasse `dvProfession` beschreibt eine Menge von Berufen. Der Datentyp `dvProfession` stellt innerhalb des JWAM Frameworks einen Fachwert (Domain Value, daher das Präfix dv) dar, der den Beruf des Versicherungsnehmers repräsentiert. Ein Fachwert ist ”sozusagen die Ergänzung zu den in jeder Programmiersprache vorhandenen Standardtypen , wie int, boolean, char und float, die sich ebenfalls nach Wertesemantik verhalten.” [BLZ99, S. 75-80] Wertesemantik bedeutet hier, dass das Objekt nicht per Referenz, sondern per Wert (Value) übergeben wird. Das JWAM Framework misst solchen ”Fachwerten”, also Objekten, die eine fachliche Bedeutung im Kontext der Anwendung haben und per Wert übergeben werden, eine große Bedeutung bei. Typischerweise sind das oft Werte die, eine String-Repräsentation besitzen. Da über eine SOAP-basierte Kommunikation prinzipiell keine Referenzen, sondern nur Werte übergeben werden können, ist das Konzept der Fachwerte für den hier betrachteten Anwendungsfall von geringer Bedeutung.

2. **Schnittstelle Vertragsverwaltungskomponente**

   - Klasse: de.jwamexample.tariffcalculator.service.ProspectAdministration
   - Interface: de.jwamexample.tariffcalculator.service.ProspectAdministrationProvider

   Die Funktionalität für die oben spezifizierten Anforderungen wird durch die Methoden `newProspect()` und `getState()` zur Verfügung gestellt. Sie besitzen folgende Signatur:

```
public Prospect newProspect()

public dvRequestState getState (dvRequestNumber rn)
```

   Die Klasse `Prospect` beschreibt die Kontaktinformationen für einen neuen Versicherungsvertrag. Ein wichtiger Wert ist der Primärschlüssel für einen solchen Auftrag, der hier als Fachwert durch die Klasse `dvRequestNumber` implementiert wird. Dieser dient als Parameter für die zweite Methode, die dann als Ergebnis den Bearbeitungsstatus der Vertragsanforderung zurückgibt. Der Rückgabewert ist ebenfalls ein Fachwert. Alle Fachwerte können, da sie über SOAP prinzipbedingt nur als Wert übertragen werden und darüber hinaus auch eine String Repräsentation besitzen, als normale Zeichenketten betrachtet werden.

Die vorgestellten internen Schnittstellen eignen sich aus mehreren Gründen nicht für eine direkte Veröffentlichung. Einerseits werden Datentypen verwendet, deren Verwendung und Umwandlung in XML unpraktisch ist. Andererseits stellen die Methoden in dieser Form nicht die gesamte Funktionalität dar, die nötig ist. Das ist am Beispiel der `newProspect()` Methode zu erkennen, die auf den ersten Blick keine Parameter über die Vertragsanforderung verlangt. Die notwendigen Informationen müssen durch den Aufruf von ”Setter”-Methoden auf dem neu erzeugten Prospekt Objekt gesetzt werden. Danach wird das Prospekt Objekt über die `updateProspect()` Methode aktualisiert. Ein weiterer Punkt betrifft die Konstruktoren der jeweiligen Klassen. Die Apache SOAP Implementierung kann nur mit Klassen umgehen, die einen ”Default”-Konstruktor (leeren Konstruktor) besitzen. Das ist bei den zwei oben aufgeführten Klassen nicht der Fall.

Darüber hinaus ist die zu exportierende Funktionalität getrennt in zwei verschiedenen Klassen implementiert, was eine Zusammenführung nötig macht, um sie als einen Dienst anbieten zu können.

Fassade

Um die oben besprochenen Probleme zu lösen, muss eine ”Fassade” geschaffen werden, die sowohl die erforderliche Funktionalität kapselt und gleichzeitig den Anforderungen der SOAP Implementierung genügt. Abb. 26 stellt den Zusammenhang der JIC Anwendung und den Web Service Erweiterungen dar. Aus den oben genannten Anforderungen an den Dienst und den von der Geschäftslogik zur Verfügung gestellten Methoden ergibt sich folgende Dienstschnittstelle:

Quellcode 5.3: ServiceInterface.java

```
1  //ServiceInterface.java
2  package wsimpl;
3
4  /*
5   This is the interface for exposed methods
6   */
7
8  import java.util.Date;
9  import de.jwamexample.tariffcalculator.material.Vehicle;
10
11 public interface ServiceInterface
12 {
13
```

```
14    public float getAnnualInsuranceRate(Vehicle vehicle,
15                            long annualDistance, int garageType,
16                            boolean isComprehensive, int liabilityRate,
17                            int comprehensiveRate, String profession);
18
19    public Vehicle[] getVehicles(String VehicleManufacturer,
20                                                              String VehicleType, int
                                                                  VehicleKw);
21
22
23    public String doEffectInsurance(Vehicle vehicle,
24                             String customerNumber, Date endOfInsuranceDate);
25
26
27    public String getInsuranceState(String rn);
28
29  }
```

1. **getAnnualInsuranceRate()**
   Berechnet die jährliche Versicherungsprämie für die Angaben, die als Parameter übergeben werden.
2. **getVehicles()**
   Liefert ein Array von Vehicleobjekten, die auf die übergebenen Parameter passen und in der Datenbank der Versicherungsgesellschaft gespeichert sind.
3. **doEffectInsurance()**
   Tätigt eine Anfrage für eine Versicherung und liefert als Rückgabewert eine Auftragsnummer zurück.
4. **getInsuranceState()**
   Liefert den Auftragsstatus für eine gegebene Auftragsnummer.

Der Quellcode der Implementation dieser Schnittstelle ist in der Datei ServiceInterfaceImpl.java abgelegt und im Anhang abgedruckt.

Da der Desktop- und der Web-Client der Anwendung über EJBs auf die Funktionalität der oben vorgestellten Implementationsklassen zugreifen und daher ohnehin ein Application Server benötigt wird, bietet es sich an, die Funktionalität der Fassaden Klasse ebenfalls über eine EJB bereitzustellen. Dies soll demonstrieren, wie über Web Services sowohl auf ”normale” Java Klassen, wie auch auf EJBs zugegriffen werden kann. Der Einsatz einer EJB ist hier aber nur als Demonstration für den zuvor genannten Punkt gedacht und ist vom funktionalen Aspekt her nicht nötig.

5.4.4 Apache SOAP Konfiguration

Die Konfiguration des SOAP Servers und das ”deployen” des Dienstes sind die zentralen Schritte, die ein Dienstanbieter tun muss, um seinen Dienst als Web Service verfügbar zu machen. Es wird auf die Installation von SOAP und auf die Konfiguration eingegangen. Zu letzterem Punkt zählt das Schreiben von Serialisieren für eigene Datentypen (z.B. Vehicle) und das Einrichten von sicherheitsbezogenen Erweiterungen. Am Ende findet das ”deployen” des Dienstes statt. Danach steht der Dienst zur Verfügung und kann via SOAP angesprochen werden.

Installation
Apache SOAP V2.2 kann unter http://xml.apache.org/soap frei heruntergeladen werden. Tomcat V3.2 kann unter http://jakarta.apache.org ebenfalls frei heruntergeladen werden. Auf die Installation von Tomact wird nicht näher eingegangen. Hier sei auf die beigelegte Dokumentation verwiesen. Für die Installation von SOAP muss das heruntergeladene Archiv zunächst entpackt werden. Danach werden die Datei soap.jar sowie die oben aufgeführten Bibliotheken in den Klassenpfad von Tomcat aufgenommen. Die server.xml datei von Tomcat wird um folgenden Eintrag erweitert:

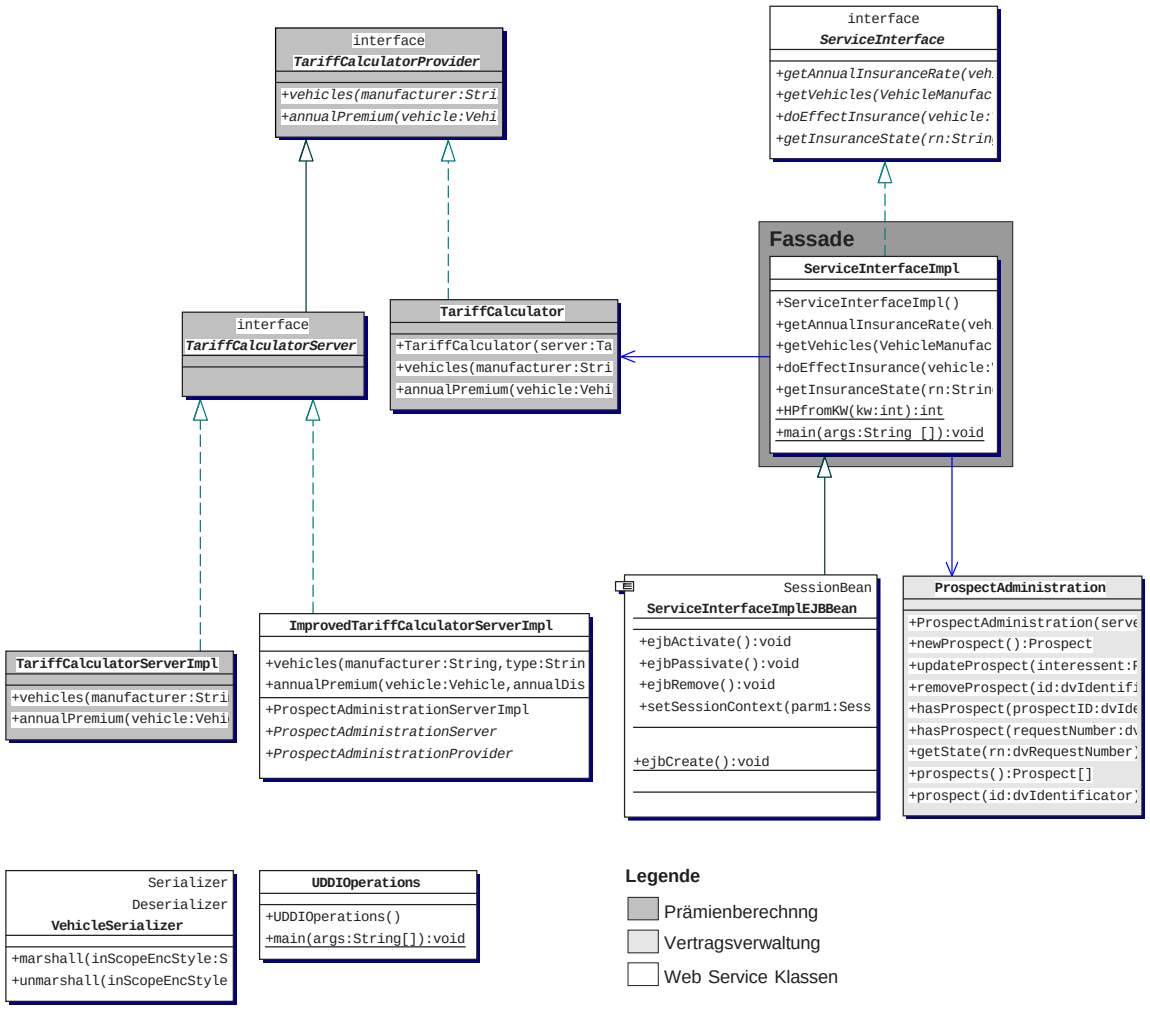

Abbildung 26: Web Service Klassendiagramm

```
<Context path="/soap"
                docBase="Pfad\soap-2_2\webapps\soap"
                crossContext="true"
                debug="0"
                reloadable="true"
                trusted="false" >
</Context>
```

Für eine genaue Installationsanweisung sei auf [CCVC01, S. 413 ff.] verwiesen.

Nach der Standardinstallation von Tomcat und SOAP müssen die folgenden Schritte durchgeführt werden:

1. **JWAM 1.6 inkl. Demos installieren**

2. **JWAM Klassen in den Klassenpfad von Tomcat aufnehmen**

3. **Tariff-Calculator Web Service Klassen in den Klassenpfad von Tomcat aufnehmen**

4. **JBoss Bibliotheken in den Klassenpfad von Tomcat aufnehmen, falls die EJB Implementation genutzt werden soll**

   - ejb.jar
   - jndi.jar
   - jboss-client.jar
   - jnp-client.jar
   - jbosssx-client.jar

5. **JBoss starten, falls die EJB Implementation genutzt werden soll**

6. **Tomcat neu starten**

Vehicle De-/Serialisierer
Der Datentyp de.jwamexample.tariffcalculator.material.Vehicle ist ein nicht von SOAP unterstützter, eigener, Datentyp. Für diesen muss ein Serialisierer/Deserialiserer geschrieben werden, der ein Objekt dieses Datentyps in eine XML Repräsentation und umgekehrt wandeln kann. Dies muss sowohl server- wie auch clientseitig geschehen. Auf der Serverseite muss unter Verwendung von Apache SOAP eine eigene Java-Klasse erstellt werden, die das Interface Serializer bzw. Deserializer aus dem org.apache.soap.util.xml Package implementiert. Diese Klasse heißt VehicleSerializer und beinhaltet die Implementation folgenden zwei Methoden, die in den oben genannten Interface spezifiziert sind:

- **Serialisieren (Java Objekt nach XML)**

```
void marshall(String inScopeEncStyle, Class javaType, Object src,
                      Object context, Writer sink, NSStack nsStack,
                      XMLJavaMappingRegistry xjmr, SOAPContext ctx)
                       throws IllegalArgumentException, IOException
```

  Die einzelnen Parameter haben folgende Bedeutung:

  – **inScopeEncStyle**
    In dieser Variablen ist die Art der Datentypenkonvertierung in/von XML festgelegt. In diesem Fall ist sie http://schemas.xmlsoap.org/soap/encoding/.
  – **javaType**
    Datentyp (Java Klasse) der serialisiert werden soll.
    Hier ist es die Klasse de.jwamexample.tariffcalculator.material.Vehicle.
  – **src**
    Das Objekt vom Typ ”javaType” welches serialisiert werden soll. In diesem Fall ist eine Instanz der Klasse de.jwamexample.tariffcalculator.material.Vehicle.
  – **context**
    Ist hier eine Objekt vom Typ java.lang.String und es enthält den Namen des XML Elementes, welches die XML Repräsentation des zu serialisierendes Objekts darstellt. Im Falle des Tariff-Calculators Web Services ist dieser Name ”item”, da ein Array übergeben wird und einzelne Arrayelemente mit ”item” gekennzeichnet werden.
  – **sink**
    Diese Variable ist eine Referenz auf den ”Writer”, mit dessen Hilfe in den XML Datenstrom geschrieben werden kann.

– **nsStack**
In dieser Variablen eines stackähnlichen Datentyps ist die Hierarchie der Namensräume der SOAP Nachricht abgelegt. Wird ein Objekt serialisiert, so muss zuerst `nsStack.pushScope()` aufgerufen werden. Damit wird ein neuer Eintrag für ein weiteres (mit einem Namensraum versehenen) Element angelegt. Nachdem das Element in den XML Datenstrom geschrieben wurde, muss `nsStack.popScope()` aufgerufen werden, um den Stackeintrag wieder zu entfernen. So kann zu jeder Zeit an jeder Position eines XML Dokumentes geprüft werden, ob ein designierter Namensraum für das aktuelle Element bereits vergeben ist.
Der Stack sieht hier folgendermaßen aus:

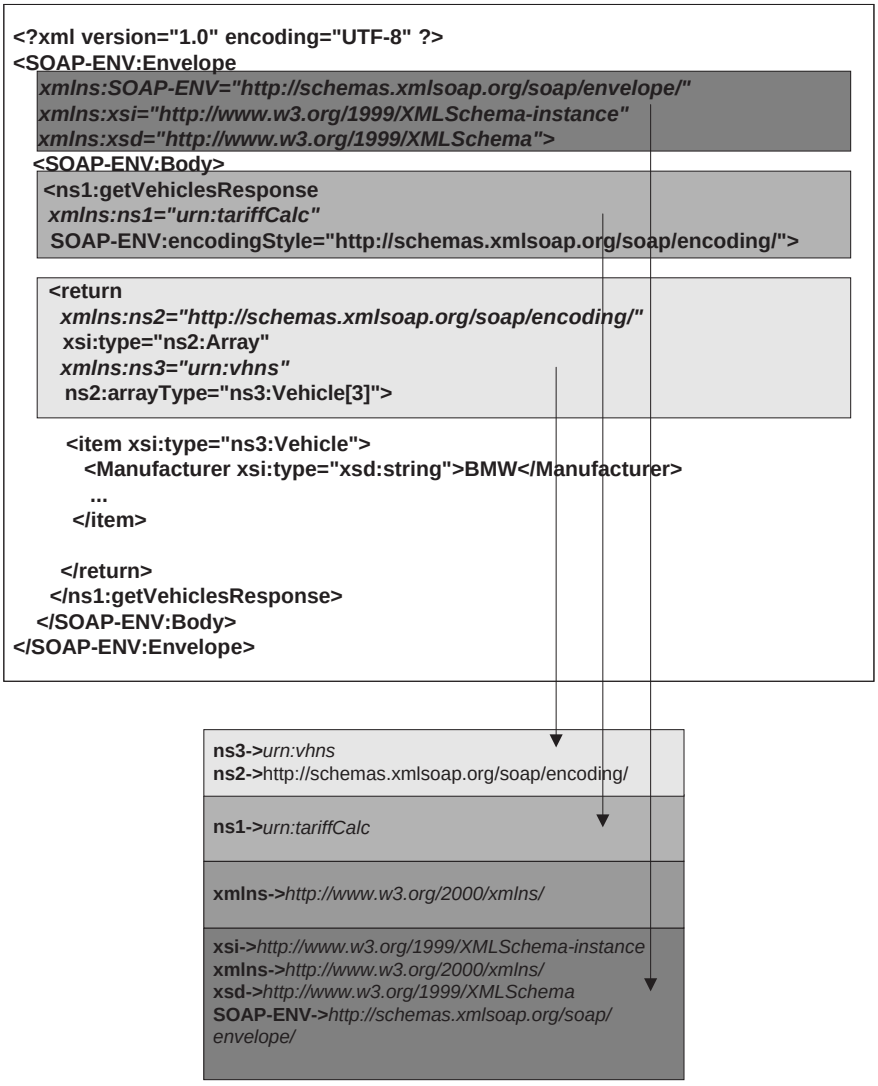

Abbildung 27: Namespace Stack einer SOAP Nachricht

– **xjmr**
Eine Referenz auf die XML Java Mapping Registry, die für die Registrierung von ”Type-Mappings” zuständig ist. Type-Mappings können entweder über den Deployment Descriptor oder durch Subklassierung der XML Java Mapping Registry festgelegt werden. Hier wird das Mapping über den Deployment Descriptor festgelegt, da dies die flexiblere Methode ist und im Falle von Änderung keine Neukompilierung erfordert.

  - **ctx**
    Ermöglich den Zugriff auf MIME-kodierte Teile der SOAP Nachricht.

- **Deserialisieren (XML nach Java Objekt)**

```
Bean unmarshall(String inScopeEncStyle, QName elementType,
                        Node src, XMLJavaMappingRegistry xjmr,
                        SOAPContext ctx)
                          throws IllegalArgumentException
```

  Die einzelnen Parameter haben folgende Bedeutung:

  - **inScopeEncStyle**
    In dieser Variablen ist die Art der Datentypenkonvertierung in/von XML festgelegt. In diesem Fall ist sie http://schemas.xmlsoap.org/soap/encoding/.
  - **elementType**
    Qualifizierter Name des Elementes, das deserialisiert werden soll.
    Hier hat es den Namen ”urn:vhns:Vehicle”.
  - **src**
    Das Wurzelelement des zu deserialisierenden Elementes als DOM Node.
  - **xjmr**
    Eine Referenz auf die XML Java Mapping Registry, die für die Registrierung von ”Type-Mappings” zuständig ist. Type-Mappings können entweder über den Deployment Descriptor oder durch Subklassierung der XML Java Mapping Registry festgelegt werden. Hier wird das Mapping über den Deployment Descriptor festgelegt, da dies die flexiblere Methode ist und im Falle von Änderung keine Neukompilierung erfordert.
  - **ctx**
    Ermöglich den Zugriff auf MIME-kodierte Teile der SOAP Nachricht.

Diese Methoden sind Callback-Methoden, die von der SOAP Implementierung aufgerufen werden, falls eine Objekt entsprechenden Typs gewandelt werden muss. Die Parameter werden also von SOAP selbst belegt und können als gegeben angesehen werden. Die Information welcher Datentyp mit welchem Serialisierer/Deserialisierer gewandelt wird, ist beim Deployment des Dienstes anzugeben. Standardtypen von Java bzw. XSD konforme Datentypen werden automatisch erkannt und gewandelt.

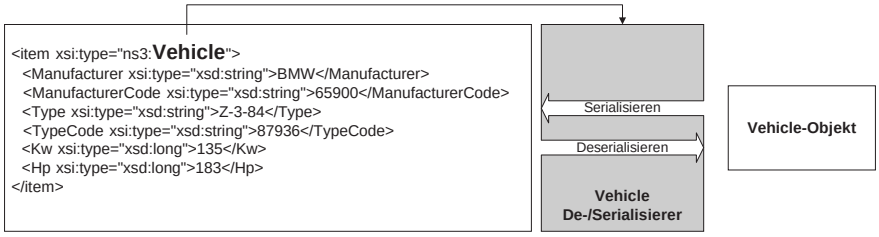

Abbildung 28: De-/Serialisierung

Deployment
Um einen Web Service nutzen zu können, muss dieser zuvor ”deployed” worden sein. Das Deployment ist das Einfügen des Services in seine Laufzeitumgebung. Es können verschiedene Informationen angegeben werden, die den Service konfigurieren und der Laufzeitumgebung die verschiedenen Parameter und Einstellungen des Services mitteilen. Diese Informationen werden im sogenannten ”Deployment Descriptor” festgelegt.

Im Falle des Tariff-Calculator Web Services sieht der Deployment Descriptor wie folgt aus:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="urn:tariffCalc">
  <isd:provider type="java"
                scope="Application"
                methods="getAnnualInsuranceRate getVehicles doEffectInsurance
                     getInsuranceState">
    <isd:java class="wsimpl.ServiceInterfaceImpl" static="false"/>
  </isd:provider>
  <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>

<isd:mappings>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   xmlns:x="urn:vhns" qname="x:Vehicle"
                   javaType="de.jwamexample.tariffcalculator.material.Vehicle"
                   java2XMLClassName="wsimpl.VehicleSerializer"
                xml2JavaClassName="wsimpl.VehicleSerializer" />
    </isd:mappings>
</isd:service>
```

- **id**
  Die eindeutige Dienstbezeichnung. Hier lautet sie `urn:tariffCalc`.
- **type**
  Angabe des Providers. In dieser Implementation ist es der mitgelieferte Provider für Java Klassen. Ist die Implementationsklasse eine EJB so wird hier ein EJB-Provider angegeben.
- **scope**
  Die Einstellung ist hier `Application`, da der Web Service wie oben beschrieben in Teilen zustandsbehaftet ist. Das instanziierte Objekt auf dem Server lebt genau so lange wie der Server selbst.
- **methods**
  Die vier Methoden die der Dienst anbietet:

  1. getAnnualInsuranceRate
  2. getVehicles
  3. doEffectInsurance
  4. getInsuranceState

  Dies sind genau die Methoden, die das Interface des Web Services zur Verfügung stellt. Im Deployment Descriptor werden sie ohne Parameter und Rückgabetyp angegeben. Diese Informationen werden zur Laufzeit bereitgestellt (z.B. über WSDL).
- **class**
  Die Klasse, welche den Dienst und damit die zuvor exportierten Methoden bereitstellt. Hier ist es die Klasse wsimpl.ServiceInterfaceImpl.

In diesem Teil des Deployment Descriptors wird das ”Type-Mapping” spezifiziert. Es wird angegeben, welche Java Klasse unter Verwendung welches Serialisierers in XML umgewandelt werden soll. Umgekehrt läßt sich ebenfalls bestimmen, mit welchen Deserialisierer ein XML Fragment in ein Java Objekt eines speziellen Types umgewandelt wird. Diese Angaben sind alle optional.

- **xmlns:x**
  Namensraum des XML Elementes. Hier ist dieser ”urn:vhns”.
- **qname**
  Name des XML Elementes. Hier ist es der Name ”x:Vehicle” (qualifiziert).

- **javaType**
  Die Java Klasse, von/in der/die gewandelt werden soll.
  In diesem Fall ist es de.jwamexample.tariffcalculator.material.Vehicle.

- **java2XMLClassName**
  Der Klassenname des Serialisierers.
  Der Serialisierer heißt hier wsimpl.VehicleSerializer.

- **xml2JavaClassName**
  Der Klassenname des Deserialisierers.
  Der Deserialisierer heißt hier ebenfalls wsimpl.VehicleSerializer.

```
java ServiceManagerClient <url> deploy <DeploymentDescriptor>
```

- **url**
  URL des SOAP Endpunktes.
  Ein Beispiel wäre http://localhost:8081/soap/servlet/rpcrouter.

- **deploymentDescriptor**
  Die Datei, welche den XML-basierten Deployment Descriptor beihnhaltet. Hier heißt die Datei DeploymentDescriptor.txt.

Der Deployment Descriptor für die EJB Variante ist im Anhang zu finden.

5.4.5 WSDL Beschreibung

Die WSDL Datei für den Web Service wird durch das Tool `wsdlgen`, das im IBM Web Service Toolkit enthalten ist, weitestgehend automatisch erstellt. Es analysiert die Signaturen der zu exportierenden Methoden und generiert ggf. XML Fragmente für zusammengesetzte Datentypen und Arrays. Bisher wird als Transportprotokoll nur HTTP unterstützt, was hier aber keine Einschränkung darstellt.

Im folgenden werden Ausschnitte der WSDL Beschreibung des Tariff-Calculator Web Services beschrieben:

- **Service**

  ```
  <service name="TariffCalcService">
     ...
     Ports
     ...
  </service>
  ```

  Das Serviceelement beinhaltet die verschiedenen Ports und legt den Namen für den Dienst fest. Hier heißt dieser TariffCalcService.

- **Port**

  ```
  <service name="TariffCalcService">
    <port name="TariffCalcServiceSoapPort"
          binding="tns:TariffCalcServiceSoapBinding">
      <soap:address location="http://localhost:8081/soap/servlet/rpcrouter"/>
    </port>
  </service>
  ```

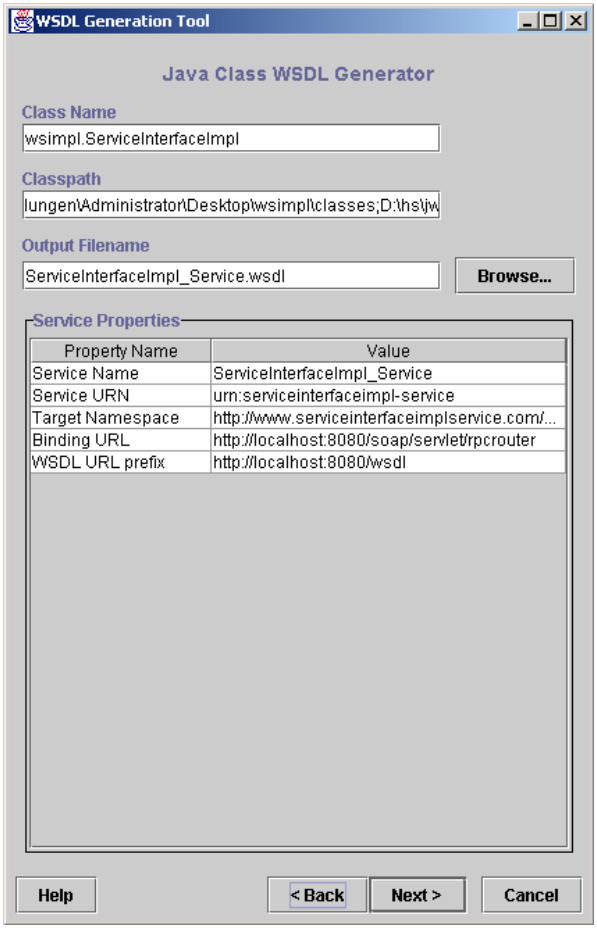

Abbildung 29: Screenshot WSTK: WSDLGEN Tool

Der TariffCalcService Dienst hat nur einen Port. Das bedeutet, dass er nur mittels einer Kombination von Transportprotokoll und Nachrichtenformat angesprochen werden kann. Das Transportprotokoll ist hier HTTP und das Nachrichtenformat ist SOAP. Diese Kombination wird in Abschnitt ”Binding” festgelegt. Zu jedem Port gehört genau eine Adresse, unter der der Dienst, mit der festgelegten Kombination aus Transportprotokoll und Nachrichtenformat, erreichbar ist.

- **Binding**

```
<binding
    name="TariffCalcServiceSoapBinding"
      type="tns:TariffCalcServiceSoapPort">
  <soap:binding
      style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>

<operation
        name="getAnnualInsuranceRate">
    <soap:operation
          soapAction="urn:tariffCalc"/>/>
    <input>
      <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:tariffCalc"
          use="encoded"/>
```

```
    </input>
    <output>
      <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:tariffCalc"
      use="encoded"/>
    </output>
  </operation>

  ...
  ...
  ...
</binding>
```

Hier wird festgelegt, dass der Dienst HTTP als Transportprotokoll und SOAP als Nachrichtenformat nutzt. Für jede Operation können Kodierungsinformationen, die je nach Nachrichtenformat variieren, angegeben werden.

- **Port Type**

```
<portType
      name="TariffCalcServiceSoapPort">

...
Operations
...
</portType>
```

Sammlung aller Operationen, die ausgeführt werden können. Ferner kann hier das Transportprimitiv festgelegt werden. In diesem Falle ist es ”Request-response” (der Standardfall).

- **Operation**

```
<portType
      name="TariffCalcServiceSoapPort">
  <operation
        name="getAnnualInsuranceRate">
    <input
        message="tns:IngetAnnualInsuranceRateRequest"/>
    <output
      message="tns:OutgetAnnualInsuranceRateResponse"/>
  </operation>
  ...
  ...
  ...
</portType>
```

Die Operationen sind eine abstrakte Beschreibung der Leistungen eines Dienstes. Hier sind es die Methoden, die aufgerufen werden können.

- **Message**

```
<message
      name="IngetAnnualInsuranceRateRequest">
  <part
      name="meth1_inType1" type="types:Vehicle"/>

  <part name="meth1_inType2"
      type="xsd:long"/>
```

```
  ...
  ...
  ...
  <part name="meth1_inType7"
    type="xsd:string"/>
</message>

<message
      name="OutgetAnnualInsuranceRateResponse">
  <part
      name="meth1_outType" type="xsd:float"/>
</message>
```

Beschreibung der Methoden mit Angabe der Datentypen für Parameter und Rückgabewerte.

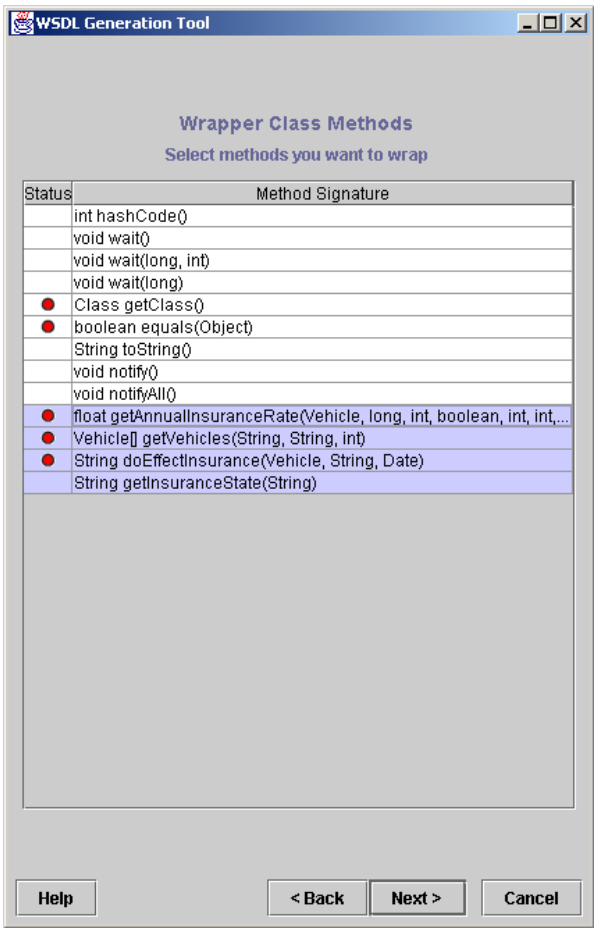

Abbildung 30: Screenshot WSTK: Automatische WSDL Generierung

- **Types**

```
<types>
  <xsd:schema
      targetNamespace="http://ServiceInterfaceImpl/types/"
      xmlns="http://www.w3.org/2001/XMLSchema/"
      elementFormDefault = "qualified"
      attributeFormDefault = "qualified">
```

```
<xsd:complexType name="Vehicle">
<xsd:sequence>
<xsd:element name="manufacturer" type="xsd:string"/>
<xsd:element name="manufacturer_code" type="xsd:string"/>
<xsd:element name="type" type="xsd:string"/>
<xsd:element name="type_code" type="xsd:string"/>
<xsd:element name="hp" type="xsd:long"/>
<xsd:element name="kw" type="xsd:long"/>
</xsd:sequence>
     </xsd:complexType>

    <xsd:complexType name="ArrayOfVehicle">
<xsd:sequence>
<xsd:element name="v" type="types:Vehicle"/>
</xsd:sequence>
   </xsd:complexType>
  </xsd:schema>
</types>
```

Festlegen von eigenen Datentypen. Hier ist es der zusammengesetzte Datentyp ”Vehicle” und dessen Variante als Array.

- **Namespaces**

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

  ...
  WSDL Content
  ...
</definitions>
```

Das ”definitions”-Tag umschließt das gesamte WSDL Dokument und beinhaltet die Definitionen für sämtliche Namensräume. Damit das WSDL Dokument von der Microsoft Implementation verarbeitet werden kann, sind hier einige kleinere Ergänzungen und Veränderungen per Hand vorzunehmen.

Besonders die Angabe, dass alle Elemente und Attribute qualifizierte Namen besitzen, ist für die Microsoft Implementierung notwendig, wird aber vom WSTK nicht berücksichtigt.

```
elementFormDefault = "qualified"
attributeFormDefault = "qualified"
```

Die Inkompatibilität erklärt sich durch die noch nicht existierende Standardisierung der WSDL Spezifikation.

5.4.6 UDDI Eintrag

Damit der Dienst gefunden werden kann, muss er in einem Verzeichnis registriert sein. Bei Web Services ist dies typischerweise ein UDDI Verzeichnis. Die Kommunikation mit dem Verzeichnis findet ebenfalls SOAP-basiert statt. Für die Registrierung des Dienstes wird die UDDI4J API von IBM benutzt, die Bestandteil des Web Service Toolkits ist. Als UDDI Verzeichnis dient das Test-UDDI Verzeichnis von IBM. Um hierauf schreibend zugreifen zu können, muss zuerst ein Zugang mit Benutzerkennung und Passwort beantragt werden. Das UDDI Verzeichnis bietet die drei folgenden Zugriffsmöglichkeiten:

1. **Web Oberfläche**
   https://www-3.ibm.com/services/uddi/testregistry/protect/registry.html

   Hier kann auch das Passwort für den schreibenden Zugriff auf das Verzeichnis beantragt werden.

2. **Publish API**
   https://www-3.ibm.com/services/uddi/testregistry/protect/publishapi

   Die sichere URL ist der Kommunikationsendpunkt, an den SOAP Nachrichten geschickt werden können, die schreibende Operationen auf dem Verzeichnis bewirken.

3. **Inquiry API**
   http://www-3.ibm.com/services/uddi/testregistry/inquiryapi

   Die URL ist der Kommunikationsendpunkt, an den SOAP Nachrichten geschickt werden können, die lesende Operationen auf dem Verzeichnis bewirken.

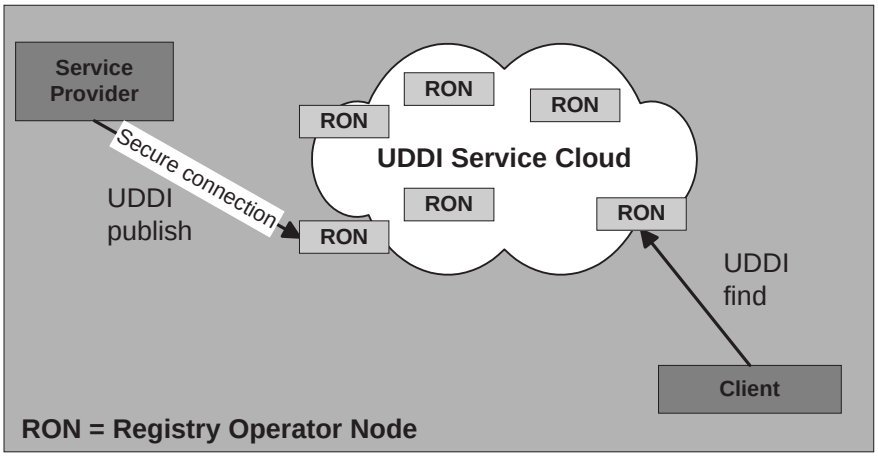

Abbildung 31: UDDI Prinzip (vgl. [CCVC01])

Mit Hilfe der UDDI API werden nun die im Kapitel 4.4.5 besprochenen Datenstrukturen aufgebaut, mit den dienstspezifischen Daten gefüllt und an das Verzeichnis geschickt. Dabei sind die Java Klassen der UDDI4J API gleich benannt, wie die entsprechende UDDI Datenstruktur.

Der Quellcodeausschnitt zeigt die Vorgehensweise:

Quellcode 5.4: UDDI4J

```
1  public class UDDIOperations {
2
3    UDDIProxy proxy = new UDDIProxy();
4
5    public UDDIOperations()
6    {
7      System.setProperty("java.protocol.handler.pkgs",
8                  "com.sun.net.ssl.internal.www.protocol");
9      Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
10
11     try
12     {
13       proxy.setPublishURL("https://www-3.ibm.com/services/uddi/ \
14                    testregistry/protect/publishapi");
15       proxy.setInquiryURL("http://www-3.ibm.com/services/uddi/ \
16                    testregistry/inquiryapi");
17
18     }catch(Exception e)
```

```
19
20      {
21        System.out.println("Exception while creating UDDI Proxy occured: "+e);
22        e.printStackTrace();
23        System.exit(-1);
24      }
25
26         Vector email = new Vector();
27      email.addElement(new Email("hendrik.saly@itelligence.de"));
28
29      Contact c = new Contact("Hendrik Saly");
30      c.setEmailVector(email);
31
32      Vector contactVector = new Vector();
33      contactVector.addElement(c);
34
35      Contacts cs = new Contacts();
36      cs.setContactVector(contactVector);
37
38      DiscoveryURL du = new DiscoveryURL("http://www.itelligence.de","http");
39      Vector duVector = new Vector();
40      duVector.addElement(du);
41
42      DiscoveryURLs durls = new DiscoveryURLs();
43      durls.setDiscoveryURLVector(duVector);
44
45      //Business Entity
46      BusinessEntity be = new BusinessEntity("","Java Insurance Company");
47      be.setAuthorizedName("Hendrik Saly");
48      be.setDefaultDescriptionString("This is the Java Insurance Company (JIC)");
49      be.setContacts(cs);
50      be.setDiscoveryURLs(durls);
51      Vector bes = new Vector();
52      bes.addElement(be);
53
54
55         try
56      {
57        AuthToken at = proxy.get_authToken("user","password");
58        System.out.println("AuthInfo: "+at.getAuthInfoString());
59
60        BusinessDetail bd = proxy.save_business(at.getAuthInfoString(), bes);
61
62
63      }catch(UDDIException e)
64
65      {
66      System.out.println("UDDI Exception: "+e);
67      e.printStackTrace();
68      }catch(SOAPException s)
69
70      {
71      System.out.println("SOAP Exception: "+s);
72      s.printStackTrace();
73      }
74    }
75  }
```

Alle Operationen auf dem UDDI Verzeichnis werden über das UDDIProxy Objekt durchgeführt. Diesem Objekt sind alle Parameter bekannt, die für einen Verbindungsaufbau nötig sind.

Danach wird die Datenstruktur aufgebaut und über die save Methoden des Proxy Objekts an das UDDI Verzeichnis übertragen.

Der vollständige Quellcode findet sich im Anhang.

5.5 Clientseitige Implementierung

5.5.1 Voraussetzungen

Die clientseitigen Voraussetzungen für den JIC Web Service unter Verwendung des MSSOAP Toolkits sind:

- **Microsoft Windows**
  Windows Betriebssystem ab Windows 98.
- **MSSOAP Toolkit V2.0 SP2**[91]
  Hierzu gehören die Bibliotheken für die SOAP Unterstützung und zwei Utilities für die Web Service Entwicklung.
- **Visual Basic 6.0 Service Pack 4**
  Visual Basic Laufzeit Bibliotheken. Das Service Pack 4 ist unter
  http://support.microsoft.com/support/kb/articles/Q235/4/20.ASP verfügbar.
- **Microsoft Excel**

5.5.2 MS SOAP Toolkit

Das Toolkit ist eine Sammlung aus Bibliotheken, die die SOAP Implementierung von Microsoft beinhalten und aus zwei Tools, mit denen SOAP Nachrichten analysiert und WSDL Dateien generiert werden können.

MS SOAP Implementierung
Die SOAP Implementierung von Microsoft besteht aus einer Sammlung von DLLs, die über jede Programmiersprache, die COM Objekte unterstützt, benutzt werden kann. Es müssen im Visual Studio folgende Verweise eingebunden werden:

- `Microsoft Soap Type Library`
- `Microsoft Soap WSDL File Generator`
- `Microsoft XML, v3.0`

Die wichtigsten Objekte sind:

- **HttpConnector**
  Ist eine konkrete Implementierung des SoapConnector Interfaces für das HTTP Transportprotokoll. Über dieses Objekt können SOAP Nachrichten via HTTP an einen Endpunkt geschickt oder von dort empfangen werden. Der HTTPConnector Objekt stellt u.a. folgende Methoden und Variablen zur Verfügung:
  - `Connector.Property()`
    Methoden, um verschiedene Eigenschaften des Connector Objekts festzulegen. Mit folgendem Aufruf wird z.B. der Kommunikationsendpunkt spezifiziert:

    ```
    Connector.Property(''EndPointUR'') =
        http://localhost:8081/soap/servlet/rpcrouter
    ```

  - `Connector.Connect()`
    Diese Methode etabliert die Verbindung mit den oben spezifizierten Eigenschaften.

[91] http://download.microsoft.com/download/xml/soap/2.0/W98NT42KMe/EN-US/SoapToolkit20.exe

  - `Connector.BeginMessage()`
    Signalisiert den Beginn einer SOAP Nachricht. Diese Methode muss aufgerufen werden, bevor die einzelnen Bestandteile einer Nachricht versendet werden können.
  - `Connector.EndMessage()`
    Signalisiert das Ende einer SOAP Nachricht und schickt diese an den festgelegten Endpunkt.
  - `Connector.InputStream`
    Variable, die den InputStream des Connectors repräsentiert. In diesen Stream kann z.B. mittels des SoapSerializer geschrieben werden.
  - `Connector.OutputStream`
    Variable, die den OutputStream des Connectors repräsentiert. Aus diesem Stream kann z.B. mit dem SoapReader gelesen werden.

- **SoapSerializer**
  Mit Hilfe von diesem Objekt wird der SOAP-Envelope generiert. Die SOAP Nachricht wird von oben nach unten aufgebaut. Das Beispiel ist an [Mic02a] angelehnt.

  ```
  Serializer.Init Connector.InputStream
  ```

  Zunächst muss der SoapSerializer mit dem InputStream des Connector Objekts initialisiert werden.

  ```
  Serializer.startEnvelope
  Serializer.startBody
  Serializer.startElement "AddNumbers", "uri:SomeURI"
  Serializer.startElement "Number1"
  Serializer.writeString "5"
  Serializer.endElement
  Serializer.startElement "Number2"
  Serializer.writeString "10"
  Serializer.endElement
  Serializer.endElement
  Serializer.endBody
  Serializer.endEnvelope
  ```

  Über die verschiedenen Methoden des Serializer-Objekts können die einzelnen Teile der SOAP Nachricht erzeugt werden.

  Die generierte SOAP Nachricht sieht wie folgt aus:

  ```
  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
   <SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Body>
        <SOAPSDK1:AddNumbers
          xmlns:SOAPSDK1="uri:SomeURI" >
           <Number1>5</Number1>
          <Number2>10</Number2>
        </SOAPSDK1:AddNumbers>
      </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
  ```

- **SoapReader**
  Mit Hilfe des SOAPReaders kann die SOAP Antwort gelesen werden. Der Reader muss zunächst mit dem OutputStream des Connector Objekts initialisiert werden.

  ```
  Reader.Load Connector.OutputStream
  ```

Danach kann über verschiedene Variablen auf die einzelnen Bestandteile der SOAP Antwort zugegriffen werden. U.a. sind dies:

- `Reader.Header`
  Referenz auf den SOAP-Header als XML-Node.
- `Reader.Body`
  Referenz auf den SOAP-Body als XML-Node.
- `Reader.RPCResult`
  Referenz auf das Ergebnis im SOAP-Body als XML-Node.
- `Reader.DOM`
  Referenz auf die DOM Repräsentation der SOAP Antwort.

SOAP Trace Utility
Mit dem MS SOAP Trace Utility kann der SOAP Nachrichtenaustausch abgehört werden. Das Utility leitet einen Port der lokalen Maschine auf einen Port einer entfernten (Remote) Maschine um und schaltet sich so in den Kommunikationsweg ein. Abb. 32 verdeutlicht das Prinzip.

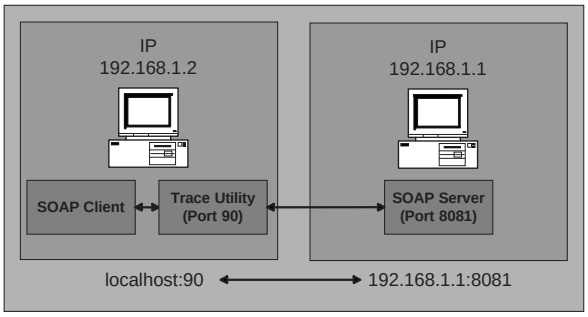

Abbildung 32: Prinzip des SOAP Trace Utility

Es kann benutzt werden, um SOAP Nachrichten sichtbar zu machen und um Fehler in der SOAP Kommunikation aufzuspüren. Ein ähnliches Programm ist auch der Apache SOAP Implementierung beigelegt. Dort heißt es TCP Monitor.

5.5.3 Visual Basic Client

Low Level Ansatz
Für den Tariff Calculation Web Service soll ein Client entwickelt werden, der den Service nutzen kann. Hier hat dieser Client ein Benutzer-Interface in Form einer Visual Basic Anwendung bzw. eines Excel Sheets. Die Funktionalität könnte Teil einer größeren Anwendung sein, auf deren Implementation im Rahmen dieser Arbeit verzichtet wurde. So ist die Berechnung der Versicherungsprämie bzw. die Vertragsabwicklung eine Teilmenge des Leasing Geschäftsprozesses der WST Bank.

Im weiteren sollen nun einige Quellcodeausschnitte erläutertet werden:

```
Private Type Vehicle
     manufacturer As String
     manufacturer_code As String
     Type As String
     type_code As String
     kw As Integer
End Type
```

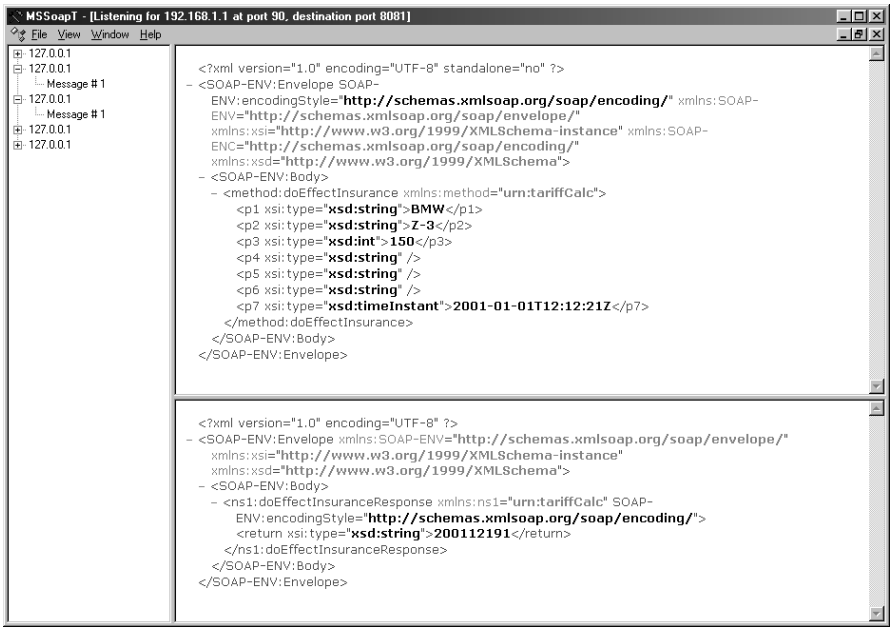

Abbildung 33: Screenshot SOAP Trace Utility

Hier wird der Datentyp `Vehicle` definiert. Dieser zusammengesetzte Datentyp entspricht der Java Klasse Vehicle auf der Serverseite.

Quellcode 5.5: Visual Basic Client

```
1  Private Function request_state(ByVal inumber As String) As String
2
3     Dim Connector As SoapConnector
4     Dim Serializer As SoapSerializer
5     Dim Reader As SoapReader
6
7     Set Connector = New HttpConnector
8     Set Serializer = New SoapSerializer
9     Set Reader = New SoapReader
10
11    Dim texts() As String
12
13    SetSoapHeader Connector, Serializer, service_id.Text, _
14                                  "getInsuranceState"
15    Serializer.startElement "p1"
16    Serializer.SoapAttribute "type", , "xsd:string", "xsi"
17    Serializer.writeString inumber
18    Serializer.endElement
19
20
21    SetSoapFooter Connector, Serializer
22    Reader.Load Connector.OutputStream
23
24
25    If Not Reader.Fault Is Nothing Then
26       MsgBox "Error occured: " & Reader.faultstring.Text, vbExclamation
27       request_state = "Error"
28    Else
29       request_state = Reader.RPCResult.Text
30    End If
31
32 End Function
```

Der Quellcodeausschnitt definiert eine Funktion `request_state`, die den Status des Versicherungsantrages

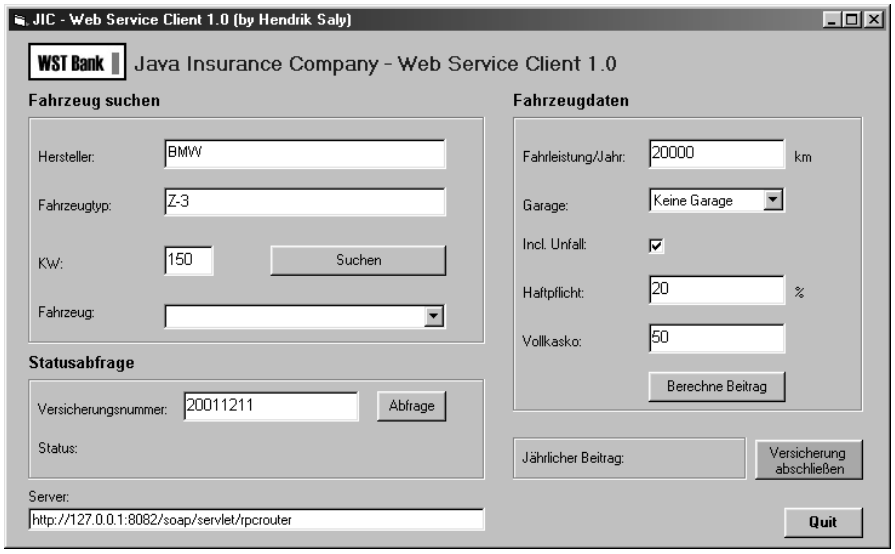

Abbildung 34: Screenshot Visual Basic Client

für eine gegebene Versicherungsnummer ermittelt. Der übergebene Parameter `inumber` wird serialisiert und als SOAP Nachricht an den Server geschickt. Die SOAP Antwort wird empfangen und ausgegeben.

Wie hier zu sehen ist, muss die gesamte Serialisierung der Datentypen vom Programmierer erledigt werden. Dieser bezieht seine Informationen aus der WSDL Beschreibung, die für den Dienst existiert. Diesem Ansatz fehlt die Dynamik, also das automatische Erzeugen von Stub Klassen, die sich um diese Aufgaben kümmern. Microsoft bezeichnet diesen Ansatz daher als Low Level Ansatz. Wie der Dienst unter Verwendung von dynamischen Binden (siehe Kapitel 4.3.3, Dynamic Binding) genutzt werden kann, ist Thema des nächsten Abschnitts.

High Level Ansatz

Der als High Level bezeichnete Ansatz zur Realisierung von Web Services mit dem MS SOAP Toolkit, bietet eine abstraktere API für den Zugriff auf Web Services, indem die komplexeren Funktionen der Low Level API verborgen werden. Als Ausgangsbasis dient eine WSDL Datei, die vom Toolkit analysiert wird. Müssen nur primitive Datentypen ausgetauscht werden, so können die entfernten Methoden direkt auf dem `SoapClient` aufgerufen werden, welches zuvor mit der WSDL Beschreibung initialisiert wurde. Es besitzt daher alle Informationen, die in der Web Service Beschreibung enthalten sind. Dies sind besonders die Adresse des Kommunikationsendpunktes und die Methodensignaturen.

Der Quellcodeausschnitt zeigt das Prinzip des High Level Ansatzes:

```
Set Client = New SoapClient
Client.mssoapinit ("http://localhost:8081/soap/TariffCalc.wsdl)

MsgBox  Client.getInsuranceState("20011201")
```

Zunächt wird ein neues Objekt vom Typ `SoapClient` erzeugt, welches dann über die Methode `mssoapinit()` mit einer WSDL Datei initialisiert wird. Danach können Methoden, die in der WSDL Beschreibung definiert sind, direkt auf dem `SoapClient`-Objekt ausgeführt werden.

In Falle des Tariff Calculator Web Services sind jedoch nicht nur primitive, sondern auch der zusammengesetzte Datentyp `Vehicle` und ein Array aus diesem Datentyp, auszutauschen. Hier stößt der High Level Ansatz bereits an seine Grenzen. Es ergaben sich folgende Probleme beim Einsatz der High Level API:

1. **Das Toolkit generiert keine `xsi:type` Datentyp Information**
   Es werden innerhalb der SOAP Nachricht keine Informationen über den jeweils zu übertragenden Datentyp geliefert. Die Apache SOAP Implementierung benötigt diese Informationen allerdings, um den entsprechenden Deserialisierer zu bestimmen und das XML-Tag korrekt deserialisieren zu können. Im oben vorgestellten Low Level Ansatz besteht dieses Problem nicht, da die Serialisierung hier in der Verantwortung des Programmierers liegt. Es gibt für diese Problematik die Lösung, dass die Namen der entsprechenden XML Elemente serverseitig beim Deployment per Hand mit den entsprechenden Deserialisierern verknüpft werden. Dies hat den Nachteil, dass nun die Namen der XML-Elemente nicht mehr verändert werden dürfen und das System so schwieriger zu warten ist.

   ```
   <!-- Deployment Descriptor -->
   ...
   <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:x=""
            qname="x:Manufacturer"
            xml2JavaClassName="org.apache.soap.encoding.
                               soapenc.StringDeserializer" />

   ...
   ```

   In diesem Beispiel wird festgelegt, dass der Inhalt des XML-Element `<Manufacturer>` eine Zeichenkette ist und mit dem Deserialisierer für Strings deserialisiert werden muss. Im Normalfall ist solch eine Festlegung nicht nötig, da das XML-Tag diese Information über das `xsi:type` Attribut selbst mitbringt:

   ```
   <Manufacturer xsi:type="xsd:string">BMW</Manufacturer>
   ```

2. **Informationen über eigene Datentypen stehen nicht dynamsich zur Verfügung**
   Es gibt keine Möglichkeit, den in der WSDL Datei beschriebenen Datentyp `Vehicle` zu benutzen. Statt dessen muss die Serialisierung/Deserialisierung auf der Clientseite auch bei der Verwendung des High Level Ansatzes selbst programmiert werden. Die Information hierfür kann natürlich der WSDL Beschreibung entnommen werden. Es zeigt sich also, dass man von einem volldynamischen automatisierten *Binding* noch weit entfernt ist.

Den Dienst finden

Wie oben erwähnt ist der Ausgangspunkt des High Level Ansatzes die WSDL Beschreibung. Diese Information kann entweder auf direktem Wege (z.B. über E-Mail oder FTP) oder über ein UDDI Verzeichnis bezogen werden. Da es mit dem Einsatz des Microsoft UDDI SDK (UDDI API von Microsoft) Installationsprobleme gab und auch der Versuch fehlschlug, den UDDI Kommunikationsendpunkt direkt über entsprechende SOAP Nachrichten anzusprechen, wurde keine UDDI Unterstützung implementiert. Das UDDI SDK benötigt als Systemvoraussetzung Windows 2000 oder höher und .NET Visual Studio. Letzteres stand nicht zur Verfügung und außerdem soll die gesamte prototypische Implementierung auch unter Windows 98/ME lauffähig sein.

Da die Kommunikation mit einem UDDI Verzeichnis SOAP-basiert ist, existiert natürlich die Möglichkeit, das Verzeichnis direkt (ohne spezielle UDDI API) über entsprechend formatierte SOAP Nachrichten anzusprechen. Jedoch scheiterte dies an einem Bug des MS SOAP Toolkits. Es kann keinen Leerstring als Wert des SOAPAction Attributes im HTTP Header übertragen. Jedoch verlangen die UDDI Kommunikationsendpunkte, sowohl von IBM wie auch von Microsoft, genau dies.

```
Connector.Property("SoapAction") = ""
```

Dieser Codeausschnitt erzeugt einen Laufzeitfehler.

Der interessierte Leser kann jedoch den folgenden Link besuchen, um im UDDI Eintrag der JIC zu blättern und alle relevanten Informationen abzurufen.

http://www.soapclient.com/uddi/uddi.sri?operator=http%3A%2F%2Fwww-3.ibm.com%2Fservices%2Fuddi%2Ftestregistry%2Finquiryapi&key=Java+Insurance+Company&requestname=find_business

Die Java Klasse `wsimpl.UDDIOperations`, welche den Code zur Veröffentlichung des Dienstes in ein UDDI Verzeichnis enthält, implementiert auch eine UDDI-Suchanfrage (Inquiry) nach der Java Insurance Company. Dies soll zeigen, dass die oben aufgezeigten Probleme Microsoft-spezifisch und nicht UDDI immanent sind.

5.5.4 Excel Client

Die oben vorgestellte Funktionalität des Web Service Client kann sehr leicht in Excel integriert werden. Hierzu muss lediglich der Quellcode aus dem Visual Studio in den Visual Basic Editor von Excel kopiert werden.

Die Vorgehensweise ist wie folgt:

1. **Neues Excel Sheet erstellen**
2. **Erforderliche Felder belegen**
3. **Erforderliche Steuerelemente generieren**
4. **Extras->Makro->Visual Basic-Editor aufrufen**
5. **Quellecode aus dem Visual Studio kopieren**
6. **Variablen von visuellen Komponenten durch Excel Zellen ersetzen**
7. **Variablen von visuellen Steuerelementen durch Excel Steuerelemente ersetzen**
8. **Über Extras->Verweise die Bibliotheken des MS SOAP Toolkits hinzufügen**

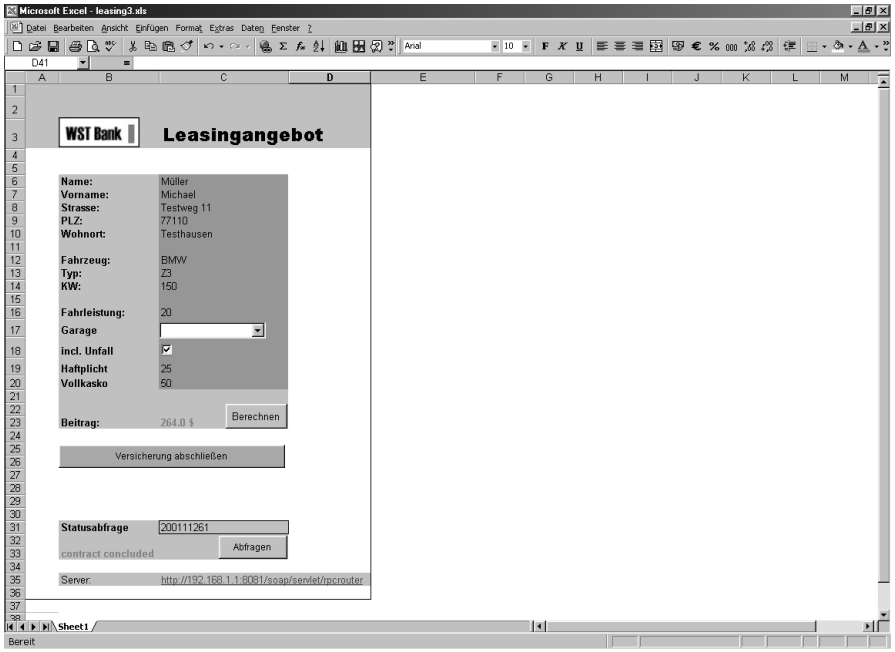

Abbildung 35: Screenshot Excel Client

Jetzt kann ein Excel-Sheet mit einer Java/EJB basierten Anwendung kommunizieren.

5.6 Bewertung

Dieses Kapitel zeigt, dass die Web Service Technologie praktisch umsetzbar ist und funktioniert. Es wird aber gleichwohl auch deutlich, dass die Technologie noch am Anfang steht und daher eine große Diskrepanz zwischen Theorie und Praxis existiert. Ferner muss berücksichtigt werden, dass zum Zeitpunkt des Schreibens ein Web Service Hype besteht und daher einige Aussagen zur Zeit lediglich Visionen oder Vorstellungen des Marketings sind. Solange nur relativ einfache Operationen mit einfachen Datentypen ausgeführt werden müssen, geht das Konzept auf. Sobald die Datentypen aber komplexer werden, zeigt sich auf der einen Seite wie unausgereift die Technologie ist und auf der anderen Seite treten Interoperabilitätsprobleme in Erscheinung, die in einfachen Anwendungen nahezu ausbleiben. Unausgereift sind vor allen Dingen die Konzepte des dynamischen Findens und Bindens. Innerhalb einer homogenen Umgebung ist dies weniger ein Problem, da hier z.B. die eigenen, komplexen Datentypen sowohl vom Client wie auch vom Server verwendet werden können. In einer heterogenen Infrastruktur hingegen, ist es zur Zeit noch notwendig, dass der Programmierer sich um die Serialisierung/Deserialisierung kümmert.

Hinsichtlich der verwendeten Implementierungen, macht Apache SOAP einen wesentlich ausgereifteren und durchdachteren Eindruck als das MS SOAP Toolkit. So sind z.B. in der Apache Distribution schon De-/Serialisierer für die gängigsten Datentypen enthalten. Das MS SOAP Toolkit unterstützt hier lediglich die Primitivtypen und Zeichenketten. Ferner leidet das Toolkit an vielen kleinen Unzulänglichkeiten, wie z.B. die oben gezeigten Probleme mit dem `xsi:type` Attribut oder dem `SoapAction` Header. Auf der anderen Seite ist Microsoft was die WSDL Verarbeitung angeht, wesentlich weiter als Apache SOAP, das praktisch keine WSDL Unterstützung bietet. Allerdings bietet die Nachfolgeimplementation Apache AXIS breite WSDL Unterstützung. AXIS befindet sich aber derzeit noch im Alpha Stadium. Besonders im Punkt WSDL macht sich die fehlende Standardisierung bemerkbar, da das vom IBM WSTK generierte WSDL noch von Hand nachbearbeitet werden musste, damit es von Microsoft Bibliotheken gelesen werden konnte.

Abschließend lässt sich sagen, dass mit Hilfe von Web Services Geschäftsprozesse zwar problemlos angestoßen werden können, aber ein Austausch komplexer Daten ist zur Zeit nur mit einem gewissen Aufwand und keinesfalls vollautomatisiert möglich.

6 Zusammenfassung und Ausblick

Dieses Kapitel fasst die wichtigsten Punkte der Arbeit zusammen und wagt einen Ausblick in die Zukunft der Web Service Technologie.

6.1 Zusammenfassung

Web Services sind eine Technologie, um die Integration von Anwendungen über das Internet zu vereinfachen. Sie sind interoperabel und erweitern die Architektur des Internets um eine dienstorientierte Komponente. Web Services sind kein Produkt, sondern eine Architektur.

Im ersten Teil der Arbeit wurde kurz erläutert was Web Services sind. Weiterhin wurde die Wahl des Themas begründet und die Ziele der Arbeit definiert. Ausführungen zur Abgrenzung des Themas beenden den ersten Teil.

Im zweiten Teil wurden die Gründe herausgearbeitet, aus denen die Web Service Technologie entwickelt wurde und eingesetzt wird. Es wurde festgestellt, dass neben wirtschaftlichen Gründen besonders der Punkt Interoperabilität eine wichtige Rolle spielt. Da Web Services durchweg auf offenen Standards aufbauen und interoperabel sind, eignen sie sich Integrationstechnologie für Anwendungen über Rechner-/Plattform- und Sprachgrenzen hinweg. Web Services weisen noch eine weitere Eigenschaft auf: Sie besitzen eine selbstbeschreibende Schnittstelle.

Im dritten Teil wurde der aktuelle Stand der Entwicklung beleuchtet. Es wurden zunächst einige Definitionen für Web Services vorgestellt und ihre wesentlichen Punkte in einer eigenen Definition zusammengefasst. Danach wurden Konzepte von verteilten Systemen beschrieben und der Versuch unternommen, die Web Service Technologie in diesen Kontext einzuordnen. Das Ergebnis dieses Versuchs ist, dass Web Services eine Technologie sind, um bereits vorhandene Technologien und Komponentenmodelle integrieren zu können. Dies wird u.a. auch im Kapitel 5 durch eine prototypische Implementierung demonstriert. Es hat sich gezeigt, dass Web Services im Prinzip nichts neues sind, sondern ”nur” eine Zusammenstellung einzelner Technologien und die durchgängige Verwendung von XML. Ferner wurden einige Frameworks und Produkte einzelner Hersteller betrachtet und gegenüber gestellt. Dabei hat sich herausgestellt, dass in diesem Punkt noch keine Entscheidung gefallen ist und es derzeit nicht abzusehen ist, wer den Markt in Zukunft dominieren wird. Am Ende des Kapitels wurden einige Adressen der wenigen zur Zeit verfügbaren Web Services genannt und die aufgetretenen Probleme diskutiert.

Im vierten Teil wurde zunächst die aktuelle Architektur des Internets beleuchtet und gezeigt, in welche Richtung sich diese Architektur in Zukunft erweitern wird. Die neue dienstorientierte Architektur wurde mit der dokumentenorientierten Architektur verglichen. Es wurde festgestellt, dass sich die Architektur wie sie heute besteht, nicht für eine Kommunikation zwischen Anwendungen untereinander, sondern nur für eine Kommunikation zwischen Mensch und Maschine eignet. Daher muss die Architektur um Teile erweitert werden, die eine Anwendungsintegration via Internet ermöglichen. Dann wurden genau diese Eigenschaften genannt und die Rollen und Vorgänge der neuen Architektur beleuchtet. Die dienstorientierte Architektur baut auf dem ”Veröffentlichen, Finden, Binden (Publish, Find, Bind)” Prinzip auf. Es sagt aus, dass ein Dienstnehmer zunächst einen Dienst suchen kann (Find) den der Dienstgeber zuvor veröffentlicht hat (Publish). Danach kann sich der Dienstnehmer an diesen Binden (Bind). Wichtig ist hier vor allem der Umstand, das durch das ”späte Binden” (Late Binding) eine hohe Flexibilität und Dynamik erreicht wird. Der Kernpunkt dieses Kapitels lag auf dem Protokollstack der Web Service Technologie. Dieser ist so aufgebaut, dass speziell formatierte XML Nachrichten über beliebige Transportprotokolle verschickt werden können. Zur Zeit ist HTTP das bevorzugte Transportprotokoll und gleichzeitig auch das einzigste, welches de-facto eingesetzt wird. Alle Daten liegen im XML Format vor. Dieser starke XML Bezug von Web Services verleiht ihnen die Interoperabilität. Der Web Service Protokollstack, besonders SOAP, definieren aber keine höherwertigen Dienste, die in einer B2B Umgebung benötigt werden. Hierzu zählen Sicherheit, Quality of Service, Transaktionen, Kontextsensitivität und die Verkettung von Web Services. Alle diese Punkte werden am Ende des Kapitels behandelt. Dabei hat sich gezeigt, dass sich zur Bereitstellung dieser additiven Mehrwertdienste noch keine Standards etabliert haben. Es existieren aber

verschiedenen Lösungsvorschläge, die ebenfalls vorgestellt wurden.

Im fünften Teil wurde das bisher theoretisch besprochene Konzept der Web Services prototypisch in die Praxis umgesetzt. Es wurde einer bestehende Anwendung der itelligence AG eine Web Service Schnittstelle hinzugefügt, die entsprechenden Beschreibungsdateien erstellt und ein Eintrag in ein UDDI Verzeichnis vorgenommen. Um den Web Service demonstrieren zu können, wurde ein Web Service Client entwickelt, der im Gegensatz zum Java-basierten Server, in Visual Basic implementiert wurde. Es hat sich gezeigt, dass Web Services für einfache Anwendungen durchaus einsatzfähig sind. Sobald die Anwendungen und besonders die zu übertragenden Datentypen aber komplexer werden, scheinen die Web Services zur Zeit noch unausgereift zu sein.

6.2 Ausblick

Diese Arbeit ist eine Momentaufnahme des aktuellen Stands der Web Service Technologie. Diese entwickelt sich mit hoher Geschwindigkeit, weshalb es sehr schwierig ist, Aussagen über Dinge wie Marktreife oder den Zeitpunkt des Durchbruchs zu machen. Web Services tragen ein großes Potential in sich, dass es in der nächsten Zeit zu entfalten gilt. Auf dem Weg dahin muss zuerst die Standardisierung der einzelnen Teiltechnologien erfolgen. Ob Web Services in dieser Ausprägung oder in einer anderen die Zukunft der dienstorientierten Architekturen sind steht nicht fest. Fest steht aber, dass sich die Idee, das Internet für dienstorientierte Zwecke zu nutzen, durchsetzen wird und einen weiteren Schritt in der Evolution des Internets darstellt.

Diese Arbeit dient der itelligence AG als Demonstrationsobjekt, um Kunden auf diese Technologie aufmerksam zu machen und ihnen die Vorteile nahezubringen. Die Implementierung wird sicherlich weiterentwickelt werden und kann als Indikator dafür dienen, wie weit die Technologie zu einem bestimmten Zeitpunkt tatsächlich ist.

A Quelltextauszüge

A.1 ServiceInterfaceImpl Klasse

Quellcode A.6: ServiceInterfaceImpl.java

```
1  package wsimpl;
2
3  /**
4   This is the implementation of the service interface.
5   It is a facade for the classes TafiffCalculator and ProspectAdministration
6   */
7
8  import java.util.Date;
9
10 import de.jwam.lang.contract.Contract;
11
12 import de.jwamexample.tariffcalculator.material.Vehicle;
13 import de.jwamexample.tariffcalculator.material.Prospect;
14
15 import de.jwamexample.tariffcalculator.service.TariffCalculator;
16 import wsimpl.ImprovedTariffCalculatorServerImpl;
17
18 import de.jwamexample.tariffcalculator.service.ProspectAdministration;
19 import de.jwamexample.tariffcalculator.service.ProspectAdministrationServer;
20 import de.jwamexample.tariffcalculator.service.ProspectAdministrationServerImpl;
21
22
23 import de.jwamexample.tariffcalculator.domainvalue.dvProfession;
24 import de.jwamexample.tariffcalculator.domainvalue.dvRequestNumber;
25 import de.jwamexample.tariffcalculator.domainvalue.dvRequestState;
26 import de.jwam.lang.domainvalue.dvInteger;
27 import de.jwam.lang.domainvalue.dvString;
28
29 import de.jwam.handling.service.ServiceProviderManager;
30
31 public class ServiceInterfaceImpl implements ServiceInterface{
32
33
34   protected TariffCalculator calculator = null;
35   protected ProspectAdministration prospectAdministration = null;
36
37   //Default Constructer
38   //Soap requires a default construcotr with no arguments
39   public ServiceInterfaceImpl()
40   {
41
42     //Startup prospect administration
43    ProspectAdministrationServer prospectServer = null;
44
45    prospectServer = new ProspectAdministrationServerImpl();// -> ergebnis wird nicht
          peristent
46
47     prospectAdministration = new ProspectAdministration(prospectServer);
48
49     Contract.ensureNotNull(prospectAdministration, "ProspectAdministration must not be
           null");
50
51     prospectAdministration.start();
52     ServiceProviderManager.instance().register(prospectAdministration);
53
54     ServiceProviderManager.instance().register(new TariffCalculator(
55          new ImprovedTariffCalculatorServerImpl()));
56
57     //Get a TariffCalculator instance
58     calculator = (TariffCalculator) ServiceProviderManager.instance().serviceProvider(
           TariffCalculator.class);
59
60     Contract.ensureNotNull(calculator, "TariffCalculator must not be null");
61   }
62
63   public float getAnnualInsuranceRate(Vehicle vehicle,
```

```
64                          long annualDistance, int garageType,
65                          boolean isComprehensive, int liabilityRate,
66                          int comprehensiveRate, String profession)
67    {
68      System.out.println(vehicle.toString()+" "+annualDistance+garageType+isComprehensive+
            liabilityRate+comprehensiveRate);
69      //At the moment we use only normal profession values
70      dvProfession dvprofession = dvProfession.Factory.instance().normalValue();
71
72      return calculator.annualPremium(vehicle,
73                          annualDistance, garageType,
74                          isComprehensive, liabilityRate,
75                          comprehensiveRate, dvprofession);
76    }
77
78    public Vehicle[] getVehicles(String VehicleManufacturer, String VehicleType,
79                          int VehicleKw)
80    {
81      return calculator.vehicles(VehicleManufacturer, VehicleType, HPfromKW(VehicleKw),
            VehicleKw);
82    }
83
84    public String doEffectInsurance(Vehicle vehicle,
85                          String customerNumber, java.util.Date endOfInsuranceDate)
86    {
87
88      //Further Implementation: Do a lookup for the customer Number an fill out the fields
            below
89
90      ProspectAdministration proxy = (ProspectAdministration)
91          ServiceProviderManager.instance().serviceProvider(ProspectAdministration.class);
92
93      proxy.start();
94      Prospect newProspect = prospectAdministration.newProspect();
95      newProspect.fieldByName("forename").fillIn(dvString.Factory.instance().value("WST"));
96      newProspect.fieldByName("surname").fillIn(dvString.Factory.instance().value("Bank"));
97      newProspect.fieldByName("title").fillIn(dvString.Factory.instance().value("End of
            Insurance: "+endOfInsuranceDate.toString()));
98      newProspect.fieldByName("street").fillIn(dvString.Factory.instance().value("
            Teststrasse 22"));
99      newProspect.fieldByName("zip").fillIn(dvInteger.Factory.instance().value("76101"));
100     newProspect.fieldByName("city").fillIn(dvString.Factory.instance().value("Musterberg")
            );
101     newProspect.fieldByName("phone").fillIn(dvString.Factory.instance().value("(032)
             11210-21"));
102     newProspect.fieldByName("fax").fillIn(dvString.Factory.instance().value("(032)
             11210-22"));
103     newProspect.fieldByName("email").fillIn(dvString.Factory.instance().value("info@wst-
            bank.de"));
104     proxy.updateProspect(newProspect);
105     proxy.stop();
106
107     return newProspect.fieldByName("requestnumber").value().toString();
108
109   }
110
111
112   public String getInsuranceState(String rn)
113   {
114     Contract.ensureNotNull(rn,this,"Requestnumber");
115     Contract.check(dvRequestNumber.Factory.instance().canCreateFromString(), "Cannot
            create requestnumber from string");
116     return prospectAdministration.getState(dvRequestNumber.Factory.instance().valueOf(rn))
            .toString();
117   }
118
119   public static int HPfromKW(int kw)
120   {
121     return (int)(1.36 * kw);
122   }
123
124   //for test purposes
125   public static void main(String [] args)
126   {
127
```

```
128     ServiceInterfaceImpl impl = new ServiceInterfaceImpl();
129     System.out.println("Web Service implementation test: ");
130     System.out.println("Annual Rate: "+impl.getAnnualInsuranceRate(new Vehicle("Mercedes",
            "SLK",HPfromKW(70),70,"assq1","dfg"),20000,1,true,20,10,"normal")+" $");
131     System.out.println("Vehicle: "+impl.getVehicles("Mercedes","SLK",200)[0]+ " (1/"+impl.
            getVehicles("Mercedes","SLK",200).length+")");
132     System.out.println("Requestnumber: "+impl.doEffectInsurance(new Vehicle("Mercedes","
            SLK",HPfromKW(70),70,"assq1","dfg"),"19001122-WSTBANK",new Date()));
133     System.out.println("Requeststate: "+ impl.getInsuranceState(impl.doEffectInsurance(new
             Vehicle("Mercedes","SLK",HPfromKW(70),70,"assq1","dfg"),"19001122-WSTBANK",new
            Date())));
134
135     int kw = 68;
136     System.out.println("FYI: "+kw +" Kw = "+HPfromKW(kw)+ " Ps");
137
138   }
139
140 }
```

A.2 EJB Deployment Descriptor

Quellcode A.7: EJB DeploymentDescriptor

```
1  <isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
2            id="urn:tariffCalcEJB">
3    <isd:provider type="org.apache.soap.providers.StatelessEJBProvider"
4              scope="Application"
5               methods="getAnnualInsuranceRate getVehicles doEffectInsurance
                    getInsuranceState">
6      <isd:option key="JNDIName" value="tariffcalculatorservice"/>
7      <isd:option key="FullHomeInterfaceName" value="wsimpl.ejb.ServiceInterfaceImplEJBHome"
           />
8      <isd:option key="ContextProviderURL" value="localhost:1098" />
9      <isd:option key="FullContextFactoryName" value="org.jnp.interfaces.
           NamingContextFactory" />
10   </isd:provider>
11   <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
12
13 <isd:mappings>
14     <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:x="urn:vhns"
           qname="x:Vehicle"
15           javaType="de.jwamexample.tariffcalculator.material.Vehicle" java2XMLClassName="
                wsimpl.VehicleSerializer"
16           xml2JavaClassName="wsimpl.VehicleSerializer" />
17
18   </isd:mappings>
19 </isd:service>
```

A.3 UDDI Operationen

Quellcode A.8: UDDIOperations.java

```
1  package wsimpl;
2
3  import com.ibm.uddi.client.*;
4  import org.apache.soap.transport.http.SOAPHTTPConnection;
5  import org.apache.soap.SOAPException;
6
7  import java.net.*;
8  import java.io.*;
9  import java.util.Vector;
```

```
10 import java.security.Security;
11
12 import com.ibm.uddi.datatype.business.*;
13 import com.ibm.uddi.datatype.service.*;
14 import com.ibm.uddi.datatype.binding.*;
15 import com.ibm.uddi.datatype.tmodel.*;
16
17 import com.ibm.uddi.datatype.*;
18 import com.ibm.uddi.response.*;
19 import com.ibm.uddi.util.*;
20 import com.ibm.uddi.*;
21
22 public class UDDIOperations {
23
24  UDDIProxy proxy = new UDDIProxy();
25
26  public UDDIOperations()
27  {
28   System.setProperty("java.protocol.handler.pkgs","com.sun.net.ssl.internal.www.protocol
        ");
29   Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
30
31   try
32   {
33    proxy.setPublishURL("https://www-3.ibm.com/services/uddi/testregistry/protect/
        publishapi");
34    proxy.setInquiryURL("http://www-3.ibm.com/services/uddi/testregistry/inquiryapi");
35
36   }catch(Exception e)
37
38   {
39    System.out.println("Exception while creating UDDI Proxy occured: "+e+"\n\n");
40    e.printStackTrace();
41    System.exit(-1);
42   }
43  }
44
45  private TModelDetail publishTModel(AuthToken at) throws SOAPException, UDDIException
46  {
47   //TModel
48   TModel tModel = new TModel("", "TariffCalculation TModel");
49   tModel.setDefaultDescriptionString("This is the Tariff Calculation TModel");
50   OverviewDoc od = new OverviewDoc();
51   od.setOverviewURL("http://info.mi.fh-offenburg.de:8037/tariffCalc.wsdl");
52   od.setDefaultDescriptionString("WSDL file");
53   tModel.setOverviewDoc(od);
54
55   Vector tmiiVector = new Vector();
56   tmiiVector.addElement(tModel);
57
58   return proxy.save_tModel(at.getAuthInfoString(), tmiiVector);
59  }
60
61
62  private ServiceDetail publishBusinessServices(AuthToken at, String BusinessEntityKey)
       throws SOAPException, UDDIException
63  {
64
65   Vector tiiVector = new Vector();
66   TModelInstanceInfo tii = new TModelInstanceInfo( ((TModel) publishTModel(at).
        getTModelVector().firstElement()).getTModelKey());
67   tii.setDefaultDescriptionString("TModel Instance Info");
68   tiiVector.addElement(tii);
69
70   TModelInstanceDetails tid = new TModelInstanceDetails();
71   tid.setTModelInstanceInfoVector(tiiVector);
72
73   //BindingTemplate
74   Vector btVector = new Vector();
75   BindingTemplate bt = new BindingTemplate();
76   bt.setBindingKey("");
77   bt.setDefaultDescriptionString("This is a SOAP over HTTP binding");
78   bt.setTModelInstanceDetails(tid);
79   bt.setAccessPoint(new AccessPoint("http://info.mi.fh-offenburg.de:8037/soap/servlet/
        rpcrouter", "http"));
80   btVector.addElement(bt);
```

```
81
82    BindingTemplates bts = new BindingTemplates();
83    bts.setBindingTemplateVector(btVector);
84
85    //BusinessService
86    Vector bsVector = new Vector();
87    BusinessService bs = new BusinessService();
88    bs.setServiceKey("");
89    bs.setBindingTemplates(bts);
90    bs.setName("TariffCalculation");
91    bs.setDefaultDescriptionString("Calculate annual tariffs for your vehicle");
92    bs.setBusinessKey(BusinessEntityKey);
93    bsVector.addElement(bs);
94
95    return proxy.save_service(at.getAuthInfoString(), bsVector);
96
97  }
98
99
100
101 private void publish()
102 {
103
104   Vector email = new Vector();
105   email.addElement(new Email("hendrik.saly@itelligence.de"));
106
107   Contact c = new Contact("Hendrik Saly");
108   c.setEmailVector(email);
109
110   Vector contactVector = new Vector();
111   contactVector.addElement(c);
112
113   Contacts cs = new Contacts();
114   cs.setContactVector(contactVector);
115
116   DiscoveryURL du = new DiscoveryURL("http://www.itelligence.de","http");
117   Vector duVector = new Vector();
118   duVector.addElement(du);
119
120   DiscoveryURLs durls = new DiscoveryURLs();
121   durls.setDiscoveryURLVector(duVector);
122
123   //Business Entity
124   BusinessEntity be = new BusinessEntity("","Java Insurance Company");
125   be.setAuthorizedName("Hendrik Saly");
126   be.setDefaultDescriptionString("This is the Java Insurance Company (JIC)");
127   be.setContacts(cs);
128   be.setDiscoveryURLs(durls);
129   Vector bes = new Vector();
130   bes.addElement(be);
131
132   try
133   {
134     //login user: jeremi1 / pass:uganda
135     //url: https://www-3.ibm.com/services/uddi/testregistry/protect/registry.html
136     AuthToken at = proxy.get_authToken("jeremi1","uganda");
137     System.out.println("AuthInfo: "+at.getAuthInfoString());
138
139     BusinessDetail bd = proxy.save_business(at.getAuthInfoString(), bes);
140     ServiceDetail sd = publishBusinessServices(at, ((BusinessEntity) bd.
          getBusinessEntityVector().firstElement()).getBusinessKey());
141
142   }catch(UDDIException e)
143
144   {
145   System.out.println("UDDI Exception: "+e+e.getFaultCode()+e.getFaultString()+e.
        getDispositionReport().getErrCode()+e.getDispositionReport().getErrInfoText());
146   e.printStackTrace();
147   }catch(SOAPException s)
148
149   {
150   System.out.println("SOAP Exception: "+s+s.getFaultCode()+s.getMessage());
151   s.printStackTrace();
152   }
153
154 }
```

```
155
156     private void find()
157     {
158
159       try
160       {
161         BusinessList bl = proxy.find_business("Java Insurance Company", null, 10);
162         BusinessInfos bi = bl.getBusinessInfos();
163         Vector biVector = bi.getBusinessInfoVector();
164
165
166         System.out.println(((BusinessInfo) biVector.get(0)).getNameString()+" /Business Key
                : "+((BusinessInfo) biVector.get(0)).getBusinessKey());
167         String businessKey =((BusinessInfo) biVector.get(0)).getBusinessKey();
168
169         BusinessDetail bd = proxy.get_businessDetail(businessKey);
170         Vector beVector = bd.getBusinessEntityVector();
171         BusinessServices bs = ((BusinessEntity) beVector.firstElement()).
                getBusinessServices();
172         Vector bsVector = bs.getBusinessServiceVector();
173
174         Vector btVector = ((BusinessService) bsVector.firstElement()).getBindingTemplates()
                .getBindingTemplateVector();
175         TModelInstanceDetails tmid = ((BindingTemplate) btVector.firstElement()).
                getTModelInstanceDetails();
176         Vector tmidVector = tmid.getTModelInstanceInfoVector();
177
178         TModelInstanceInfo tii = (TModelInstanceInfo) tmidVector.firstElement();
179         TModelDetail tmd = proxy.get_tModelDetail(tii.getTModelKey());
180         Vector tmVector = tmd.getTModelVector();
181         TModel tm = (TModel) tmVector.firstElement();
182
183         System.out.print("WSDL file: "+tm.getOverviewDoc().getOverviewURLString());
184
185
186       }catch(UDDIException e)
187       {
188         System.out.println("UDDI Exception: "+e+e.getFaultCode()+e.getFaultString()+e.
                getDispositionReport().getErrCode()+e.getDispositionReport().getErrInfoText());
189         e.printStackTrace();
190       }catch(SOAPException s)
191       {
192         System.out.println("SOAP Exception: "+s+s.getFaultCode()+s.getMessage());
193         s.printStackTrace();
194       }
195     }
196
197
198   public static void main(String[] args)
199   {
200
201     UDDIOperations op = new UDDIOperations();
202
203     //Uncomment to publish
204     //op.publish();
205
206     op.find();
207
208   }
209 }
```

B Literatur

Literatur

[Ada01a] ADAM, Dr. C. *Home of the Web Services Community*. http://www.webservices.org/whatarewebservices.php. 23.09.2001

[Ada01b] ADAM, Dr. C. *Home of the Web Services Community*. http://www.webservices.org/modules.php?op=modload&name=FAQ &file=index&myfaq=yes&id_cat=2&categories=Webservices+v0.4#7. 23.09.2001

[All01] ALLAIRE CORPORATION. *WDDX FAQ, Resources, and Web Sites*. http://www.allaire.com/handlers/index.cfm?ID=5624&Method=full. 18.11.2001

[Apa01] APACHE. *Apache SOAP v2.2 Documentation*. http://xml.apache.org/soap/docs/. 18.11.2001

[BEA01] BEA SYSTEMS. *Datasheet - BEA WebLogic Server 6.1)*. http://www.bea.com/products/weblogic/server/datasheet.shtml. 04.12.2001

[BHK+01] BOUBEZ, Toufic ; HONDO, Maryann ; KURT, Chris ; RODRIGUEZ, Jared ; ROGERS, Daniel. *UDDI Data Structure Reference V1.0*. http://www.uddi.org/pubs/DataStructure-V1.00-Open-20000930_2.pdf. 18.11.2001

[BHL01] BRAY, Tim ; HOLLANDER, Dave ; LAYMAN, Andrew. *Namespaces in XML*. http://www.w3.org/TR/REC-xml-names/. 18.11.2001

[BLZ99] BLEEK, Wolf-Gideon ; LILIENTHAL, Carola ; ZÜLLIGHOVEN, Heinz: Frameworkbasierte Anwendungsentwicklung (Teil 4): Fachwerte. In: *OBJEKTspektrum* 5 (1999), September/Oktober

[BP01] BANSAL, Sonal ; PAL, Gaurav. *The Web at your (machine's) service*. http://www.javaworld.com/javaworld/jw-09-2001/jw-0928-smsservice.html. 18.11.2001

[BPSMM01] BRAY, Tim ; PAOLI, Jean ; SPERBERG-MCQUEEN, C. M. ; MALER, Eve. *Extensible Markup Language (XML) 1.0 (Second Edition)*. http://www.w3.org/TR/REC-xml. 13.11.2001

[CCMW01] CHRISTENSEN, Erik ; CURBERA, Francisco ; MEREDITH, Greg ; WEERAWARANA, Sanjiva. *Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl. 18.11.2001

[CCVC01] CAULDWELL, Patrick ; CHAWLA, Rajesh ; VIVEK CHOPRA, u.a.: *Professional XML Web Services*. Wrox Press, 2001

[Cem01] CEMPER, Christoph C. *Context Propagation, oder: das Session Handling bei Web Services*. Vortragsunterlagen - WebServices Konferenz Stuttgart (29-31.10.2001). 13.11.2001

[Cos01] COSTELLO, Roger L. *XML Schemas: Best Practices*. http://www.xfront.com/BestPracticesHomepage.html. 18.11.2001

[Cov01a] COVER, Robin. *The XML Cover Pages - Universal Description, Discovery, and Integration (UDDI)*. http://www.oasis-open.org/cover/uddi.html. 07.12.2001

[Cov01b] COVER, Robin. *The XML Cover Pages - Web Services Flow Language (WSFL)*. http://www.oasis-open.org/cover/wsfl.html. 18.11.2001

[Du01] DU, Dr. G. *Transaction model of Web Services and its impact on application Design*. Vortragsunterlagen - WebServices Konferenz Stuttgart (29-31.10.2001). 13.11.2001

[Ele01] ELECTRIC, Mind. *UDDI*. http://www.themindelectric.com/products/glue/releases/GLUE-1.2/docs/glue/guide/uddi.html. 18.11.2001

[Ga94] GAMMA, Erich ; ET AL.: *Design Patterns- Elements of Reusable Object-Oriented Sofware*. Addison-Wesley, 1994

[GHMN01] GUDGIN, Martin ; HADLEY, Marc ; MOREAU, Jean-Jacques ; NIELSEN, Henrik F. *SOAP Version 1.2 - W3C Working Draft 2 October 2001*. http://www.w3.org/TR/soap12-part1/, http://www.w3.org/TR/soap12-part2/. 25.09.2001

[GT00] GRUHN, Volker ; THIEL, Andreas: *Komponentenmodelle*. 1. Addison-Wesley Verlag, 2000

[Haa01] HAASE, Kim. *JMS Tutorial*. http://java.sun.com/products/jms/tutorial/1_3-fcs/doc/basics.html. 25.09.2001

[Han01] HANSEN, Sven. *Microsoft stoppt Passport-Wallet-Service*. http://www.heise.de/newsticker/result.xhtml?url=/newsticker/data/sha-03.11.01-003/default.shtml&words=Passport. 11.11.2001

[Hol01] HOLUBECK, Andreas: Alles unter einem Dach. In: *Java Magazin* 9 (2001), September

[IBM01a] IBM. *IBM Software: Solutions: Web services by IBM: Documentation*. http://www-4.ibm.com/software/solutions/webservices/documentation.html. 08.12.2001

[IBM01b] IBM. *Web Services and UDDI*. http://www-3.ibm.com/services/uddi/standard.html. 18.11.2001

[Jec01a] JECKLE, Mario. *Die besondere Rolle von XML im EAI*. http://www.jeckle.de/files/eai2001.pdf. 13.11.2001

[Jec01b] JECKLE, Mario. *Making Machines Talk Together – Systemintegration mit XML-Sprachen*. http://www.jeckle.de/files/InformatikTagZweibruecken_20001124.pdf. 13.11.2001

[Jec01c] JECKLE, Mario. *XML-Tutorial*. http://www.jeckle.de/vorlesung/xml/script.html. 13.11.2001

[Jec01d] JECKLE, Mario. *SOAP - aber sicher!* http://www.jeckle.de/files/sicheresSOAP.pdf. 18.11.2001

[Jew01] JEWELL, Tyler. *The Business Transaction Protocol*. http://www.sys-con.com/webservices/article2a.cfm?id=55&count=11213&tot=10&page=8. 07.12.2001

[Knu01] KNUTH, Michael: Web Services - die Middleware der Zukunft? In: *Java Magazin* 9 (2001), September

[Kre01] KREGER, Heather. *Web Services Conceptual Architecture*. http://www-4.ibm.com/software/solutions/webservices/pdf/WSCA.pdf. Mai 2001

[Kü01] KÜHNE, Anderas. *Business Transactions*. Vortragsunterlagen - WebServices Konferenz Stuttgart (29-31.10.2001). 13.11.2001

[Lai01] LAIFER, Roland. *Architektur von DCE*. http://www.dce.de/architektur.html. 24.09.2001

[Ley01a] LEY, Michael. *Database Systems & Logic Programming*. http://www.informatik.uni-trier.de/ ley/d-bi/ein.html. 18.11.2001

[Ley01b] LEYMANN, Prof. Dr. F. *Web Services Flow Language*. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf. 18.11.2001

[Ma00] MARTIN, Didier ; ET AL.: *Professional XML*. Wrox Press, 2000

[Mac01] MACK, Iris. *Einführung in DCE*. http://www.uni-stuttgart.de/rus/Bi/1997/9+10/file4.html. 24.09.2001

[Mat01] MATTERN, Friedemann. *Informatik-Spektrum, Abstract Volume 24 Issue 3 (2001) pp 145-147, Springer-Verlag Berlin Heidelberg 2001*. http://link.springer.de/link/service/journals/00287/bibs/1024003/10240145.htm. 23.09.2001

[Meg01] MEGGINSON, David. *SAX 2.0: The Simple API for XML*. www.megginson.com/SAX. 13.11.2001

[Mer01] MERKLE, Bernhard. *(R)evolution - IIOP: Alternative zu HTTP*. http://www.heise.de/ix/artikel/1997/07/136/. 13.11.2001

[MH01] MONSON-HAEFEL, Richard: *Enterprise Java Beans*. 2. O'Reilly Verlag, Köln, 2001

[Mic01a] MICROSOFT CORPORATION. *XML/Web services/.NET White Papers - Microsoft Service Providers*. http://www.microsoft.com/serviceproviders/whitepapers/xml.asp. 10.11.2001

[Mic01b] MICROSOFT CORPORATION. *XML Web Services*. http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000442. 23.09.2001

[Mic02a] MICROSOFT. *SOAP Toolkit 2.0 Documentation*. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/soap/htm/soap_ref_soapconnector_9er7.asp. 05.01.2002

[Mic02b] MICROSOFT CORPORATION. *Microsoft .NET Framework - Technical Overview*. http://www.gotdotnet.com/team/framework/DotNet Framework Technical Overview v3.doc. 05.01.2002

[Mod01] MODI, Tarak. *Clean up your wire protocol with SOAP, Part 1*. http://www.javaworld.com/javaworld/jw-03-2001/jw-0330-soap.html. 18.11.2001

[Nel01] NELSON, Chris. *IBM Webcast - Question 61*. http://webevents.broadcast.com/ibm/developer/062701/display_qa.asp. 11.11.2001

[OAS02] OASIS. *OASIS Technical Committees - Business-Transactions Committee*. http://www.oasis-open.org/committees/business-transactions. 05.01.2002

[OK01] ORAM, Andy ; KAN, Gene: *Peer-to-Peer Harnessing the Power of Disruptive Technologies*. 1. O'Reilly Verlag, 2001

[PA01] PLUMMER, Daryl ; ANDREWS, Whit. *The Hype Is Right: Web Services Will Deliver Immediate Benefits*. http://www3.gartner.com/DisplayDocument?doc_cd=101571. 06.11.2001

[Pie01] PIEFEL, Michael. *CORBA-Einführung*. http://www.informatik.hu-berlin.de/ corsica/Seminar/WS97/CORBA/CORBA-InterOp.html. 11.11.2001

[Pro01] PROF. DR. GREGOR ENGELS. *Web Engineering*. http://www.uni-paderborn.de/cs/ag-engels/ag_dt/Courses/Lehrveranstaltungen/SS01/WE/Begleitunterlagen/Kap5-bw-6up.pdf. 10.11.2001

[Pü01] PÜRCKHAUER, Christoph. *Service Oriented Architecture - Webservices im EAI and B2B Umfeld*. Vortragsunterlagen - WebServices Konferenz Stuttgart (29-31.10.2001). 13.11.2001

[R. 01] R. FIELDING AND ET AL. *RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1*. http://www.w3c.org/Protocols/rfc2616/rfc2616.txt. 10.11.2001

[Rei01] REICHARDT, Diana. *A Field Guide to Services On Demand and Sun ONE*. http://www.sun.com/software/sunone/wp-fieldguide/wp-fieldguide.pdf. 18.11.2001

[Rie01] RIETHE, Andreas. *DB-Aspekte des E-Commerce, Schwerpunkt: Techniken*. http://wwwdbis.informatik.uni-kl.de/courses/seminar/SS2001/ausarbeitung4.pdf. 18.11.2001

[Rol01] ROLLER, Dieter. *Business Processes and Web Services*. http://www.it.fht-esslingen.de/ñonnast/IT-Forum/WebServices_Roller.pdf. 13.11.2001

[RW02] ROOCK, Stefan ; WOLF, Henning. *JWAM Javadoc Dokumentation*. http://www.jwam.de/product/product_doc/api/. 05.01.2002

[Sar01] SARAKATSANIS, Athanasios: Einmaleins der Web Services. In: *Java Magazin* 9 (2001), September

[Sch01] SCHADLER, Ted: CTOs: Wake Up To Web Services. In: *The Forrester Brief* (19.07.2001)

[See01] SEEMANN, Michael: Spürhunde im Web - Web Service Architektur: Das dynamische eBusiness. In: *Java Magazin* 6 (2001), Juni

[Shi01] SHINKA AG. *Shinka Solution Services*. http://www.shinka.de/solution/solution_success.phtml. 18.11.2001

[Smi01] SMITH, David. *Microsoft Continues Web Service Leadership With New XML Specs*. http://www3.gartner.com/DisplayDocument?doc_cd=98366. 18.11.2001

[Spi01] SPILIOPOULOU, Myra. *Grundlagen von verteilten Systemen, Humboldt-Universität zu Berlin*. http://www.wiwi.hu-berlin.de/ myra/VSvorlesung/Einfuehrung.html. 23.09.2001

[SS00] SAAKE, Gunter ; SATTLER, Kai-Uwe: *Datenbanken and Java. JDBC, SQLJ and ODMG.* dpunkt-Verlag, Heidelberg, 2000

[Sta01] STAHL, Michael: Stets zu Diensten: Web-Services im Überblick. In: *OBJEKTspektrum* 4 (2001), Juli/August

[Sun01a] SUN MICROSYSTEMS. *Jini Network Technology FAQs*. http://www.sun.com/jini/faqs/. 04.10.2001

[Sun01b] SUN MICROSYSTEMS. *Web Services*. http://java.sun.com/j2ee/webservices/, http://dcb.sun.com/practices/webservices/. 10.11.2001

[Sun01c] SUN MICROSYSTEMS. *Sun Open Net Environment FAQ*. http://www.sun.com/software/sunone/faq.html#2. 23.09.2001

[TPC01] TODD, Stephen ; PARR, Francis ; CONNER, Michael H. *A Primer for HTTPR*. http://www-106.ibm.com/developerworks/webservices/library/ws-phtt/. 18.11.2001

[UDD01] UDDI. *Offzielle UDDI Webseite*. http://www.uddi.org. 18.11.2001

[VJ01] VAWTER, Chad ; JUNE, Ed R. *J2EE vs. Microsoft.NET*. http://www.theserverside.com/resources/article.jsp?l=J2EE-vs-DOTNET. 11.11.2001

[W3C01a] W3C. *Document Object Model (DOM)*. http://www.w3.org/DOM/. 13.11.2001

[W3C01b] W3C. *XML Schema*. http://www.w3.org/XML/Schema. 18.11.2001

[Wey01a] WEYER, Christian. *Plattformübergreifende Objektkommunikation im Internet über XML mit SOAP*. http://www.eyesoft.de/download/presentation/SOAP_OOP2001.pdf. 08.12.2001

[Wey01b] WEYER, Christian: Öffentlicher Dienst: Entwicklung von ".NET Webservices". In: *OBJEKTspektrum* 4 (2001), Juli/August

[Wie01] WIELAND, Dr. T. *Programmieren mit UDDI*. Vortragsunterlagen - WebServices Konferenz Stuttgart (29-31.10.2001). 13.11.2001

[Wol01] WOLFF, Eberhard. *J2EE and Webservices*. Vortragsunterlagen - WebServices Konferenz Stuttgart (29-31.10.2001). 13.11.2001

[XAM01] XAML. *XAML - Transaction Authority Markup Language*. http://www.xaml.org. 18.11.2001

[Zü98] ZÜLLIGHOVEN, Heinz: *Das objektorientierte Konstruktionshandbuch. Nach dem Werkzeug- und Materialansatz.* dpunkt-Verlag, Heidelberg, 1998