Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Institut für Praktische Informatik und Medieninformatik
Fachgebiet Telematik

Betreuer: Thorsten Strufe

Hauptseminar Web Services
im SS 2003

Einführungsvortrag

Hendrik Rusch
Langewiesener Str. 4
98693 Ilmenau
Hendrik.Rusch@stud.tu-ilmenau.de

Matrikel-Nr. 29085

Inhalt

Einleitung

Im vergangenen Jahrzehnt ist die weltweite Rechnervernetzung ständig vorangeschritten. Lokale und globale Netze, mit dem Internet als prominentestes Beispiel, entwickelten sich zu einem wichtigen und viel genutzten Medium für Nutzer auf aller Welt. Dies trifft für private Haushalte ebenso zu wie für Unternehmen. Gerade im Unternehmensbereich brachte die Vernetzung neue Möglichkeiten der Interaktion. Sowohl E-Business, die elektronische Abwicklung von Geschäftsprozessen, als auch e-Commerce, der Verkauf über elektronische Medien, haben in den letzten Jahren zunehmend an Bedeutung gewonnen und sind ein wichtiger Bestandteil der Unternehmenswelt.

Im Bereich des e-Commerce bieten Unternehmen bis heute ihre Dienstleistungen dem Kunden meist über Internetseiten an. Die Entwicklung dynamischer Inhalte schaffte dafür weitere, neue Möglichkeiten der Interaktion. Jedoch beschränkt sich der Nutzerkreis, abgesehen von spezialisierten Programmen, auf natürliche Personen.
In Zukunft sollen die Möglichkeiten im Bereich des B2C (Business-to-Customer) ausgebaut werden. Neben der Standardisierung der Kommunikation mit dem Nutzer dreht es sich um die Schaffung neuer Kommunikationsmöglichkeiten. So sollen auch Geräte eines Kunden Dienstleistungen nutzen können. Vom einfachen Abfragen einer Postleitzahl bis hin zum Kühlschrank, der automatisch Lebensmittel bestellt, sind viele Anwendungen denkbar.
Dies bedeutet, dass die unterschiedlichsten Applikationen auf diversen Plattformen miteinander kommunizieren müssen. Natürlich bedarf es dafür einer Architektur, welche eine den Anforderungen genügende Zusammenarbeit der verteilten Systeme anbietet. Der Web Service Technologie wird zugetraut, dies umsetzen zu können.

Das Gebiet der Interaktion von Unternehmen und Kunden ist jedoch nur ein Teil der Anforderungen im Bereich der vernetzten Systeme. Sowohl in als auch zwischen Unternehmen ist die Zusammenarbeit verschiedenster Anwendungen ein ebenso wichtiger Aspekt. Diverse Geschäftsanwendungen müssen miteinander kommunizieren und zusammenarbeiten, um effizient Geschäftsprozesse auszuführen.
Innerhalb von Unternehmen müssen die unterschiedlichsten Geschäftsbereiche miteinander interagieren. Das Thema EAI (Enterprise Application Integration), das Verbinden verschiedener Geschäftsanwendungen im Unternehmen, ist ein aufwendiges Anwendungsfeld. Vorhandene und neue Anwendungen in den verschiedenen Bereichen sollen effizient

miteinander zusammenarbeiten. Dazu gehören einfacher Zugriff auf Funktionalitäten, problemlose Kooperation der Applikationen und einfache Erweiterbarkeit. Das derzeitige Problem: sind Systeme verschiedener Hersteller im Einsatz, müssen für funktionsfähige Zusammenarbeit z.T. aufwendige Daten- oder Protokollumwandlungen durchgeführt werden. Eine standardisierte, einfache und flexible Infrastruktur soll dies vereinfachen. Und dort sollen Web Services einhaken. Diese sollen ermöglichen, dass die unterschiedlichsten Anwendungen, Datenbanken etc. einfach und problemlos miteinander kommunizieren können. Der Einsatz von Legacy-Systemen auch in neuen Umgebungen soll unterstützt werden. Und auch Techniken für den Workflow im Unternehmen, ein Teilbereich der EAI, sind bereits in Entwicklung.

Ein weiterer großer Anwendungsbereich ist das B2B (Business-to-Business), die Zusammenarbeit und Kooperation von Unternehmen. Auch begünstigt durch viele Kooperationen und Fusionen in den letzten Jahren, müssen die unterschiedlichsten Transaktionen zwischen Unternehmen, Märkten und der Industrie bewältigt werden. Und dies weltweit, mit teils wechselnden und neuen Partnern

Die Ausführung der dafür benötigten Geschäftsprozesse ist in steigender Form auf die Interaktion des Backends der logistischen Kette angewiesen – seien dies Computer-Systeme, Softwareanwendungen oder Geschäftsapplikationen.

Die beteiligten Anwendungen sollten aber nicht eng gekoppelt sein, was heute noch oft der Fall ist. Zu viele Vereinbarungen und Voraussetzungen führen dazu, dass sich nur eigens zugeschnittene Programme verstehen. Zur Kommunikation herstellerfremder Produkte, bei Umstieg auf einen neuen Partners oder Erneuerung eigener Systeme müssen oft extra Schnittstellen erstellt werden. Es sollen sich aber Systeme praktisch aller Hersteller mit wenig Aufwand verstehen. Am besten ohne menschlichen Eingriff. Automatisiertes e-Business stellt eines der Ziele dar. Und letztendlich soll es möglich sein, benötigte Dienste dynamisch bei Partnern oder anderen Anbietern zu finden und zu nutzen. Anonymisierte, erst zur Laufzeit gefundene Dienste sollen Bestandteil der zukünftigen Anwendungswelt werden. Und dies alles sollte auch die Integration vorhandener Anwendungen durch einfaches Aufsetzen eines Frontends unterstützen.

Um diese Interaktionen zu ermöglichen wird eine standardisierte, flexible Infrastruktur für den Datenaustausch in e-Business Anwendungen benötigt. Just-in-time Integration steht dabei ganz oben auf der Anforderungsliste. Auch hier sollen Web Services eine Lösung darstellen.

Umsetzungen für den Bereich der verteilten Anwendungen, auch für die erwähnten Anforderungen, gibt es aber schon länger. Die verschiedensten Hersteller entwickelten Techniken wie CORBA, RMI oder DCOM. Diese hatten aber mehrere Nachteile. Zum einen ist sicherlich das Problem der zu engen Kopplung der verschiedenen beteiligten Komponenten zu nennen. Diese Kopplung konnte sich z.B. durch vorgegebene Programmiersprache (RMI), festgelegtes Betriebssystem (DCOM) oder auch ein bestimmtes Objektmodell (CORBA) äußern. Fehlende Unabhängigkeit von Plattform, Programmiersprache oder zugrunde liegenden Protokollen war eine unangenehme Folge. Dem kommt als schwerwiegender Punkt hinzu, dass diese Techniken meist auch den Einsatz neuer Protokolle, Compiler oder sonstige technischen Voraussetzungen erforderten. Zusätzliche Neuinvestitionen waren die Folge, oder aber es wurde auf den Einsatz der Technik verzichtet. Und ein weiterer Aspekt, der nicht zu unterschätzen ist, betrifft die Komplexität der Entwicklung. Eine hohe Einstiegsbarriere und großer Entwicklungsaufwand halten Unternehmen oftmals von Neuentwicklungen ab.
Zusammen mit der fehlenden Einigung der Industrie auf einen gemeinsamen Standard sind dies Gründe, die den umfassenden Erfolg der Techniken versagten.
Durch die längere Präsenz auf dem Markt haben sie aber schon Marktanteile für sich gewonnen. In Bereichen, wo die benötigte Infrastruktur und Techniken vorhanden sind oder speziell zugeschnittene Anwendungen entwickelt wurden werden sie daher in nächster Zukunft selten durch neue Technologien verdrängt werden.

Was ist ein Web Service?

Angesichts des (teils abgeebbten) Hypes um Web Services mag es einfach erscheinen, eine eindeutige Definition zu finden. Jedoch fällt dies in der Fülle des vorhandenen Materials, das oft widersprüchlich ist oder nur Teile umfasst, ziemlich schwer.

Die W3C Web Services Architecture Working Group konnte sich auf folgende Beschreibung festlegen[1]:
„A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols.“

[1] W3C /Web Services Architecture/

Einfach gesagt ist ein Web Service eine Sammlung von Funktionen, die in einer Einheit verpackt sind und für die Nutzung in einem Netzwerk veröffentlicht werden.[2] Er bildet damit eine einzelne Komponente. Seine Funktionen können, ebenso wie die Art des Zugriffs und die Lokalisierung des Dienstes, in XML-basierten Dateien beschrieben werden. Diese stellen die Dienstbeschreibung dar. Klienten können zur Laufzeit den Web Service entsprechend der benötigten Funktionalität ausmachen, z.B. in dafür vorgesehenen Verzeichnissen. Anhand seiner Dienstbeschreibung kann der Klient die bereitgestellte Funktionalität über ein Netzwerk für sich nutzen. Die dafür nötige Kommunikation findet per Übermittlung XML-basierter Nachrichten statt. Durch die Trennung der Dienstbeschreibung von der Implementierung kann der Dienst selbst dann noch wie gewohnt benutzt werden, wenn sich etwas an der Implementierung geändert ist. Wichtig ist nur, dass die bisher in der Dienstbescheibung vorhandenen Funktionen auch weiterhin unterstützt werden, d.h. die Schnittstelle gleich bleibt.

Bei den Klienten handelt es sich ebenfalls um Anwendungen, die auch selbst Web Services sein können. Durch die Interaktionen bildet sich ein Netz von lose gekoppelten Komponenten.

Web Services sind für maximale Interoperabilität über das Internet konzipiert. Wichtigstes Argument für Web Services ist somit sicher, dass ihre Architektur es Programmen ermöglicht, unabhängig von der implementierenden Programmiersprache, Netzwerkprotokollen und verwendeten Plattformen miteinander zu kommunizieren. Zudem können Web Services dynamisch zur Laufzeit gefunden und eingebunden werden, um Funktionalitäten in eigene Anwendung zu integrieren. Somit lassen sich Applikationen einfach mit austauschbaren, zum Entwicklungszeitpunkt unbekannten, Komponenten entwickeln.

Ermöglicht werden sollen diese Eigenschaften durch die Festlegung auf Standards. Der Einsatz von XML und Standardprotokollen, die meist auf XML basieren, vereinigt eben jene geforderte Interoperabilität.
XML wird zum einen bei der Entwicklung von Web Services benutzt. Sie werden nach gleichen Standards für die Selbstbeschreibung, Publikation, Lokalisierung, Aufruf, Kommunikation und Datenaustausch entwickelt. Die daraus entstehende Dienstbeschreibung, die durch XML-Dokumente realisiert wird, kann von allen Anwendungen gelesen und für die

[2] Glass /WS (r)evolution/

Interaktion mit einem Web Service genutzt werden. Standard hier ist das Format WSDL. Da die Beschreibung unabhängig von der eigentlichen Implementierung ist, sind verwendete Programmiersprache und Plattform freigestellt.
Zudem werden für die Kommunikation über das Netzwerk XML-basierte Nachrichten verwendet. Die Nachrichten sind unabhängig vom Transportprotokoll und von der Art der Übermittlung. SOAP hat sich jedoch als quasi-Standard für das Transportprotokoll etabliert. Web Services arbeiten somit hervorragend mit standardisierten Internet-Protokollen, wie HTTP und TCP/IP zusammen. Eine entsprechende Web-Infrastruktur ist in vielen Unternehmen auch schon vorhanden, somit können oftmals Neuinvestitionen entfallen.
Und auch die Verzeichnisse, in denen Dienste angeboten und gefunden werden, nutzen XML zur Beschreibung der vorhandenen Dienste.

Zu den Eigenschaften gehören somit[3]

- Plattformunabhängigkeit
- Protokollunabhängigkeit
- Unabhängig von der Programmiersprache
- Transport über Standard-Internetprotokolle
- Dokumenten- und Nachrichtenbasierte Kommunikation
- XML-Basierung

Durch breite Unterstützung in der Industrie ist die Akzeptanz von Web Services sowohl unter Anbietern als auch unter Nutzern ziemlich sicher.

Um es noch einmal zu verdeutlichen: es gibt keinerlei Vorschriften bezüglich der verwendeten Technik für Implementierung, Datentransport, Lokalisierung etc. SOAP oder WSDL haben sich zwar als de-facto Standards etabliert, sind aber keinesfalls verbindlich anzuwenden.
Um dies zu verdeutlichen gehe ich zuerst auf die den Web Services zugrunde liegende Architektur ein.

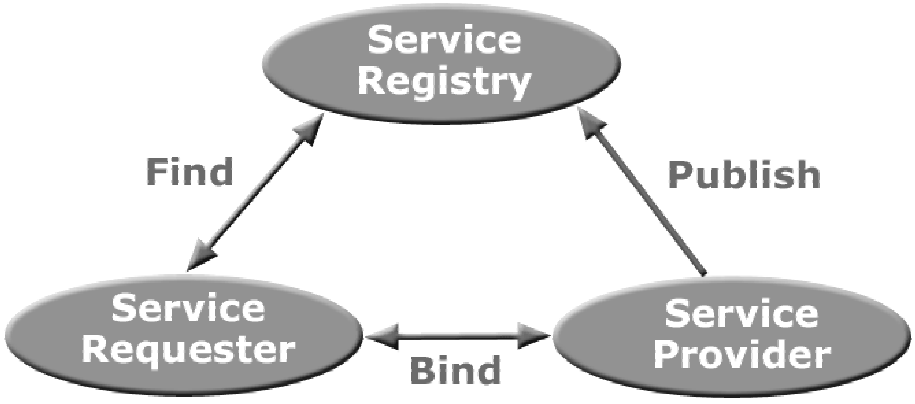
[3] Lau /Developing Web Services/

Das Architekturkonzept[4]

Die den Web Services zugrunde liegende Architektur ist die so genannte Service-Oriented Architecture (SOA). Ist die lose Kopplung bzw. Integration von Anwendungen gefordert, lässt sich die SOA einsetzen. Sie ist damit auch gut für die Implementierung von e-Business geeignet.

Die SOA ist keine neue Erfindung. Schon vor ein paar Jahren erschien HP mit einer Implementierung von SOA, hatte jedoch wegen den proprietären Anforderungen keinen maßgeblichen Erfolg. Jetzt jedoch, nachdem einige große Unternehmen wie IBM oder Microsoft die Vorzüge des Konzepts entdeckt haben, scheint ein Erfolg um einiges wahrscheinlicher. Sie erkannten den entscheidenden Punkt dafür: die Implementierung muss auf offenen Standards basieren. Diese werde ich später, im Abschnitt „Technische Architektur“, noch vorstellen – u.a. SOAP und WSDL.

Unabhängig von der Implementierung umfasst die SOA 3 Rollen. Diese führen wiederum 3 Interaktionen aus. Die Rollen sind dabei der Service Provider, der Service Requester sowie die Service Registry. Diese interagieren miteinander und führen dabei die Aktionen Publish, Find und Bind aus.



Rollen und Interaktionen der SOA

[4] iX /Web-Programmierung/
Gottschalk /WS Architecture/
Gisolfi /dynamic e-business/

Die Rollen

Der **Service Provider** ist der eigentliche Besitzer bzw. Anbieter des Dienstes. Er hat diesen wahrscheinlich in einer beliebigen Programmiersprache entwickelt und besitzt eine Beschreibung des Dienstes im XML-Format. Er kann den Dienst durch verschiedene Möglichkeiten einem potentiellen Nutzer anbieten. Die Art und Weise, wie dann vom Nutzer auf den Dienst zugegriffen wird, bleibt dabei dem Provider überlassen. Jedoch hat sich SOAP für die Datenübertragung als Quasi-Standard durchgesetzt[5].

Der **Service Requester** stellt den Dienstnutzer dar. Er führt eine Suchoperation aus, um einen gewünschten Dienst zu finden und dessen Beschreibung zu erlangen. Durch die Beschreibung erfährt er, wie der Dienst zu nutzen ist. Es kann sich beim Requester sowohl um eine natürliche Person als auch um ein Stück Software handeln. Ersterer würde den Dienst z.B. auf einer Webseite finden, die auch die zugehörige Beschreibung enthält, und ihn dann nutzen, d.h. direkt beim Anbieter anfordern. Letztere dagegen würde die Aufgabe automatisch ausführen und den Dienst womöglich in einem entsprechenden Verzeichnis finden. Aufgrund der dort enthaltenen Dienstbeschreibung erhält die Anwendung die zur Interaktion benötigten Informationen. Hat sie den benötigten Dienst gefunden stellt sie eine Verbindung zu diesem her und nutzt ihn.
Betrachtet man die Kommunikation als Server-Client-Beziehung, so stellt der Requester den Client dar. Die Software kann sowohl lokal liegen (im Falle eines Intranet) als auch entfernt vorhanden sein (Internet).

Die **Service Registry** stellt eine Art Gelbe Seiten für Web Services dar. Genauer handelt es sich um zentrale Verzeichnisse und Bibliotheken.
Hier kann der Provider die benötigten Informationen über sich sowie die von ihm zur Verfügung gestellten Dienste eintragen. Daten über die Art und Weise des Aufrufs wie auch die Verwendung werden ebenso eingetragen. Durch die Möglichkeit der Klassifizierung der Dienste und der Suche innerhalb der Register kann der Nutzer die verschiedenen Dienste bewerten und untereinander vergleichen. Hat er einen geeigneten Dienst gefunden kann er diesen dynamisch binden.

[5] iX /Web-Programmierung/

Wie schon angedeutet beinhaltet das Szenario bestimmte Interaktionen. Diese werden in 3 Operationen als Bestandteil der SOA gegliedert. Im nächsten Abschnitt werden diese näher betrachtet.

Die Interaktionen

Das **Veröffentlichen** (Publish).
Dies ist jede Aktion eines Providers, die eine Dienstbeschreibung für seinen angebotenen Dienst dem potentiellen Requester verfügbar macht. Es stehen verschiedene Möglichkeiten des Veröffentlichens zur Verfügung, abhängig davon, wie dynamisch die Einbindung des Dienstes gedacht ist.
Am einfachsten ist die direkte Veröffentlichung. Hierbei stellt der Provider die Dienstbeschreibung dem Requester direkt zur Verfügung. Denkbar sind dafür das Versenden als Anhang einer E-Mail, das Publizieren auf einer FTP-Seite oder auch auf einem Datenträger.
Etwas dynamischer ist die Nutzung von DISCO oder ADS. Es werden über einfache HTTP GET Aufrufe die Dienstbeschreibungen von einer URL abgefragt.
Dem Konzept der SOA entspricht die Einbeziehung einer Service Registry, welche volle dynamische Dienstsuche unterstützt. Umsetzungen der Service Registry, wie z.B. UDDI, speichern die Dienstbeschreibungen und bieten zum Teil umfangreiche Suchmöglichkeiten.

Das **Finden** (Find).
Jede Aktion, die einem Requester Zugang zur Beschreibung für einen Dienst verschafft, wird als Finden bezeichnet. Im einfachsten Fall kann dies ein Zugriff auf eine Datei oder eine URL sein, welche die Beschreibung enthält. Andernfalls handelt es sich im Normalfall um einen Zugriff auf eine Service Registry. Dies geschieht durch eine XML-Anfrage an diese Registry. Dort sind die Dienste nach verschiedenen Kategorien abgelegt, und es finden sich Verweise auf die Dienstbeschreibung. In den Verzeichnissen lässt sich nach verschiedenen Kriterien wie Qualitätsaspekte, vom Provider unterstützte Protokolle oder Kategorie suchen.

Das **Binden** (Bind).
Das Binden erzeugt die eigentliche Client/Server-Beziehung, die Kommunikation des Requester mit dem Provider. Einfachste Möglichkeit ist das statische Binden. Kennt der

Requester bereits einen gewünschten Dienst und weiß wie dieser anzusprechen ist, kann er die direkte Kommunikation über z.B. über SOAP initialisieren.
Findet er einen Dienst erst zur Laufzeit ist dies nicht möglich. Durch die Suche in einer Registry erhält er aber eine dem Dienst zugehörige Dienstbeschreibung. Der Requester nutzt nun die in der Beschreibung angegebenen Protokolle und Schnittstellen, um den Dienst ausfindig zu machen, zu kontaktieren und zu erwecken. Dafür kann zur Laufzeit mittels der erhaltenen Informationen ein Proxy generiert werden, welcher die Kommunikation durchführt.

Die technische Architektur

Um Web Services entsprechend den Anforderungen entwickeln und einsetzen zu können braucht man geeignete Technologien. Die technische Struktur der Web Services ist ebenenförmig im so genannten Web Services Protocoll Stack aufgebaut. Dies ist eine Sammlung von standardisierten Protokollen und APIs, welche Personen und Anwendungen die Möglichkeit geben, Web Services zu finden und zu nutzen. Jede Ebene des Stacks bietet einfache, offene Protokolle und APIs[6]. Dies soll zum einen die Interoperabilität und eine erfolgreiche Implementierung der SOA sichern, aber auch die Akzeptanz und den Erfolg in der Unternehmenswelt gewähren.

Die Ebenen des Protocoll Stack

[6] Gottschalk /WS Architecture/

Die Grundlage des Stack stellt ein **Netzwerk** dar. Jeder Web Service, der im Sinne der SOA eingesetzt werden soll, muss über ein Netzwerk verfügbar sein. Über dieses Netzwerk finden die Kommunikation eines Nutzers mit dem gewünschten Web Service statt, aber auch die Anfragen an Service Registries. Für die Transportprotokolle des Netzwerkes bestehen dabei vielfältige Möglichkeiten. Neben der Übertragung per HTTP, welche sicherlich den Status als Standard genießt, sind auch andere Protokolle wie SMTP oder FTP denkbar.

Darauf folgt eine XML-basierte **Nachrichten-Schicht**. Diese ermöglicht die Kommunikation zwischen dem Web Service und seinen Clients sowie Anwendungen mit Service Registries. Der Datenaustausch findet meist mittels SOAP statt, welches sich als Quasi-Standard etabliert hat.
Über diese Nachrichten-Schicht findet der Austausch von Nachrichten statt, der benötigt wird, um Web Services zu publizieren, zu finden und zu nutzen. Dadurch werden somit die Publish-, Find- und Bind-Aktionen der SOA ermöglicht.

Die 3. Schicht stellt die **Service Description** dar. Mittels der Dienstbeschreibung werden alle Informationen festgehalten, die zur Nutzung eines Dienstes benötigt werden. Dazu gehören die öffentlichen Funktionen inkl. auszutauschender Nachrichten und Datentypen, aber auch Adressinformationen zur Lokalisierung des Dienstes. Als de-facto Standard hat sich hier WSDL etabliert.

Die aufgezeigten unteren 3 Ebenen sind mindestens vonnöten, um interoperable Web Services zu unterstützen[7].
Um das gesamte Konzept der SOA umzusetzen muss aber auch die Komponente Service Registry implementiert werden. Dies kommt im Stack durch die Ebenen Service Publication und Service Discovery zum Ausdruck. Diese Ebenen stellen den Vorgang der Veröffentlichung und des Auffindens von Diensten dar, aber auch die Speicherung der zur Verfügung gestellten Informationen. Hier hat sich UDDI als Standard herausgebildet, worauf auch später noch eingegangen wird.

Neben den dargestellten Ebenen kann der Stack nach oben hin noch erweitert werden. Diese Schichten sind jedoch optional und können entsprechend den Geschäftsbedürfnissen genutzt werden.

[7] Gottschalk /WS Architecture/

Bei der Erklärung des Stacks sind schon verschiedene zugrunde liegende Technologien erwähnt worden. Im Folgenden werden diese näher betrachtet.

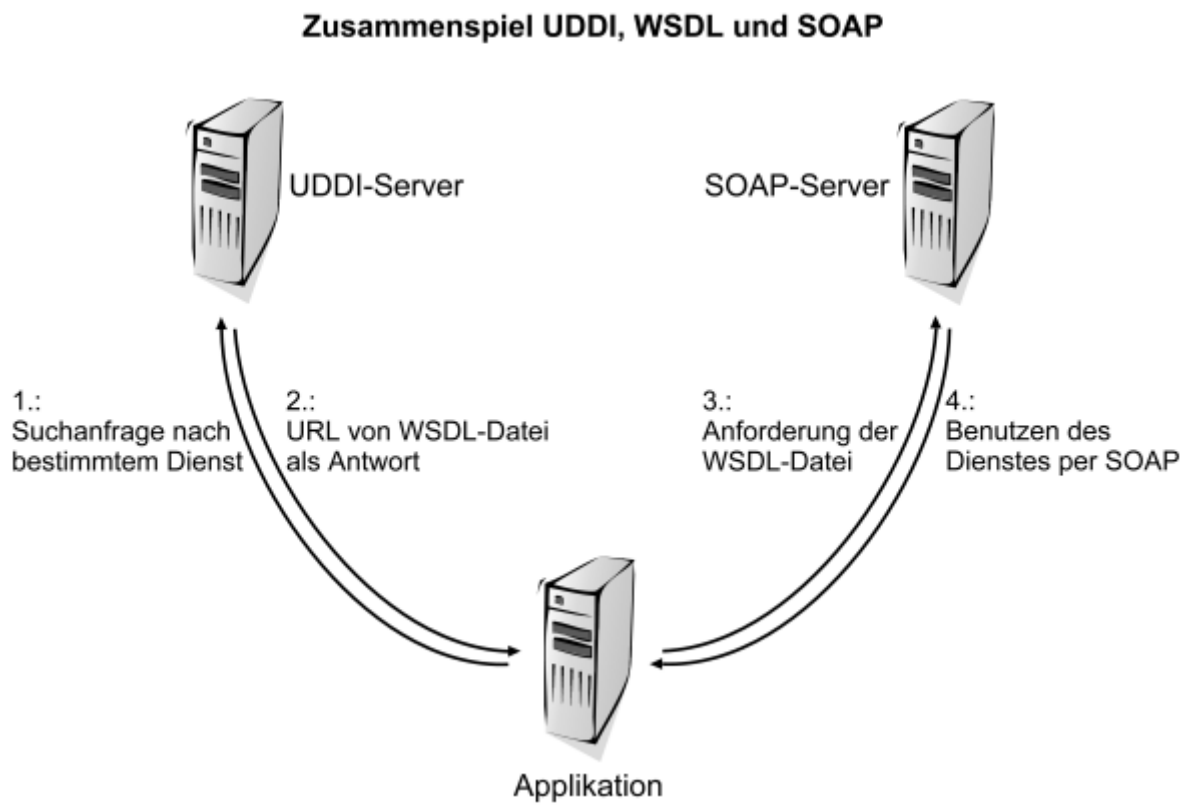
XML Schema[8]

Wie schon erwähnt basieren fast alle Standards, die zur Erstellung und Nutzung von Web Services verwendet werden, auf XML.
XML-Dokumente haben immer eine bestimmte Struktur, die Aufbau und verwendete Elemente des Dokuments angibt. Für die Sicherung des Verständnisses und die Wahrung der Struktur auf Ersteller- und Nutzerseite sollten XML-Dokumente in irgendeiner Form definiert werden. Zu diesem Zweck wurde XML Schema entwickelt. Es dient der Definition der Struktur und Typen der in einem Dokument enthaltenen Elemente und Attribute. Der Vorteil von XML Schema gegenüber einer älteren Technik, den DTD: sie enthaltenen bereits Definitionen von einfachen programmiersprachlichen Typen. Diese können bei der Beschreibung der Nachrichten für die Kommunikation zwischen Diensten oder für die Schnittstellenbeschreibung problemlos verwendet werden. Die Zuordnung von Daten zu Datentypen lässt sich leicht erledigen, wobei neben den enthaltenen auch eigene komplexe Typen verwendet werden können.
XML Schema ist selbst in XML beschreiben, wodurch es mit XML Parsern verarbeitbar ist.

XML Namespace

Jedes XML-Dokument beinhaltet diverse Elemente und Attribute, welche entsprechende eigene, meist selbst zugewiesene Namen haben. Damit diese selbst bei der Kombination verschiedener Dokumente oder XML-Schemas unterschiedlicher Herkunft eindeutig sind gibt es zusätzliche Namenskonventionen. Element- und Attributnamen setzen sich danach aus einer URI und einem lokalen Namen zusammen. Durch die URI kann die Eindeutigkeit des gewählten Namens auch über Organisationsgrenzen hinweg gesichert werden.
Die URI wird durch eine Namensraum-Definition an ein Namensraum-Präfix gebunden. Die erfolgt durch die Angabe xmlns:*prefix=URI*. Das so definierte Präfix kann dann im Dokument nach dem Prinzip <prefix:localName>…</...> verwendet werden.

[8] Stuart /XML-Schema/



Die Techniken im Zusammenspiel [Horn /SOAP, WSDL, UDDI/]

SOAP

Für die Kommunikation der SOA-zugehörigen Rollen wird ein entsprechendes Nachrichten-Übertragungsprotokoll benötigt. Dies sollte unabhängig von Plattformen nutzbar sein und mit verschiedenen Transportprotokollen zusammenarbeiten.

SOAP (Simple Object Access Protocol) ist ein leichtgewichtiges, XML-basiertes Protokoll, entwickelt für den Datenaustausch in dezentralisierten, verteilten Umgebungen[9]. XML-basiert, da es die zu übermittelnden Nachrichten in ein XML-Dokument verpackt. SOAP ist für den Einsatz mit verschiedenen Transportprotokollen geeignet, wird aber bevorzugt mit HTTP verwendet. Denn HTTP erfreut sich weiter Verbreitung und ist auch über Firewalls hinweg einfach einsetzbar. Es wäre aber auch z.B. der Versand per E-Mail möglich.
Positive Eigenschaften von SOAP sind sicher die Unabhängigkeit von Plattformen, Programmiersprachen, Betriebssystem und Objektmodell. Zudem ist SOAP erweiterbar, d.h.

[9] Gisolfi /dynamic e-business/

neue Funktionalität kann der Übertragung hinzugefügt werden. Dies ist z.B. für das Einfügen von Sicherheitsfunktionen verwendbar.
SOAP lässt sich in zwei Kommunikationsmodellen einsetzen. Zum einen für die dokumentenbasierte Übertragung, wobei beliebige Dokumente übermittelt werden können. Darauf aufbauend lassen sich Remote Procedute Calls realisieren. Hier stellt eine XML-Nachricht mit bestimmtem Format eine Anfrage dar, worauf eine Antwortnachricht mit Ergebnissen und Fehlermeldungen folgt.

Eine SOAP Nachricht besteht immer aus einem so genannten SOAP-Umschlag (envelope). Dieser enthält die Elemente Header und Body. Ersterer ist optional und enthält anwendungs-spezifische Steuerungsinformationen. Letzterer enthält nachrichtenorientiert das zu übertragende Dokument oder, im Falle des RPC, den eigentlichen Methodenaufruf mit Parametern bzw. eine Antwort mit Ergebnissen. Es können am Ende der SOAP-Nachricht auch noch weitere Daten angehängt werden, z.B. für Sicherheitsfunktionen oder Transaktionssteuerung.

Ein Nachteil von SOAP hängt eng mit den eigentlichen Vorteilen zusammen. Interoperabilität. lässt sich durch die Verwendung von XML gut realisieren. Jedoch zieht XML einen relativ großen Overhead mit sich. Gerade bei der Kommunikation über Bandbreiten-begrenzte Netzwerke kann dies Probleme bereiten. Auch der Einsatz für zeitkritische Übertragungen z.B. im Multimedia-Bereich ist somit nicht unbedingt die Stärke von SOAP.
Zudem ist die SOAP-Spezifikation ziemlich komplex und somit für Entwickler nicht schnell zu erfassen. Jedoch können Entwicklungstools hier Abhilfe schaffen.

Trotz der Nachteile hat SOAP sich auf dem Markt durchgesetzt, nicht zuletzt durch die breite Herstellerunterstützung.
Einzig wahre Alternative scheint XML-RPC zu sein. Diesem fehlen neben der genannten Unterstützung aber der eingebaute Erweiterungsmechanismus und einige weitere Features. Trotzdem könnte es für kleinere Anwendungen oder Testeinsätze eine sinnvolle Alternative darstellen.[10]

WSDL

[10] Webservices.XML.com /SOAP/

Wurde ein Dienst erstellt, muss dieser in irgendeiner Form beschrieben werden. Nur so wird es anderen Anwendungen ermöglicht, diesen ohne spezielles Vorwissen zu nutzen. Insbesondere müssen seine Funktionen mit dazugehörigen Aufrufkonventionen erklärt sein. Für diese Dienstbeschreibung wurde die Sprache WSDL entwickelt.

Eigenschaften

WSDL (Web Services Description Language) ist eine XML-basierte Sprache. Dies bietet den Vorteil, dass sie sowohl von Personen als auch von Anwendungen lesbar ist. Erstere können dies nutzen, wenn sie eine Anwendung für einen bestimmten, ihnen bekannten Dienst entwickeln möchten, für welchen sie schon ein WSDL-Dokument besitzen. Anwendungen dagegen können durch die Beschreibung mit dynamisch gefundenen Diensten kommunizieren. Mit WSDL lassen sich die Web Services unabhängig von der gewählten Implementierungssprache beschreiben. WSDL beschreibt im Einzelnen die öffentlichen Funktionen und Datentypen des Web Service, das Mapping zwischen den Parametern und den Datentypen für das Protokoll und das Binding des Web Service an die unterliegenden Transportprotokolle[11].
Man kann WSDL damit als einen Vertrag ansehen, der zwischen Anbieter und Nutzer eines Web Service abgeschlossen wird. Dem entsprechend muss der Anbieter gewährleisten, dass der Dienst entsprechend der Beschreibung nutzbar ist.

Technisch gesehen beschreibt WSDL Netzwerkdienste als eine Reihe von Endpunkten, die auf Nachrichten operieren, welche nachrichtenorientierte oder prozedurorientierte Informationen enthalten. Die Nachrichten und Operationen werden abstrakt beschrieben, um dann an ein konkretes Netzwerkprotokoll und Nachrichtenformat gebunden zu werden. Damit wird ein Endpunkt festgelegt, wobei in Beziehung stehende konkrete Endpunkte in abstrakte Endpunkte kombiniert werden[12].
WSDL ist erweiterbar. Damit können Endpunkte und ihre Nachrichten unabhängig von Netzwerkprotokollen und Nachrichtenformaten beschrieben werden. Zurzeit sind jedoch nur Bindings für SOAP, HTTP POST und MIME beschrieben[13].

[11] iX /Web-Programmierung/
[12] W3C /WSDL/
[13] W3C /WS Architecture Description/

WSDL-Dokument und WSDL-Daten

Das WSDL-Dokument lässt sich in 2 Bereiche gliedern: Die Service Interface Definition, der abstrakte Teil, und die Service Implementation Definition, der konkrete Teil. Die Interface Definition enthält Angaben darüber, was der Dienst tut und wie ein Aufruf formatiert werden muss, damit er vom Anbieter verarbeitet wird. Die Implementation Definition beschreibt, wo genau sich die Implementierung des Dienstes befindet, d.h. über welche protokollspezifische Netzwerkadresse er aufrufbar ist[14].

Was gehört zu den abstrakten Definitionen?[15]

Types sind maschinen- und sprachenunabhängige Typdefinitionen, welche in den ausgetauschten Nachrichten verwendet werden. Wenn der Dienst nur die einfachen Typen benutzt, die im XML Schema beinhaltet sind, braucht kein types-Element angelegt zu werden.

Messages sind abstrakte Beschreibungen der ausgetauschten Daten, die sowohl vom als auch zum Web Service übertragen werden. Jedes message-Element beschreibt eine Nachricht. Es legt den Namen der Nachricht und mittels enthaltenen Part-Elementen Parameter oder Rückgabewerte fest. Zu **PortType** werden verschiedene Operationen eines Web Service hinzugefügt. Operationen legen fest, welche XML-Nachrichten als Input- und Outputnachrichten auftreten können. Ein PortType kann z.B. eine Request- und eine Response-Nachricht in eine einzelne Request/Response-Operation kombinieren.

Die konkreten Beschreibungen sind wie folgt:

Ein konkretes Protokoll und Datenformat für die Operationen und Nachrichten, die in einem PortType definiert sind, bilden ein wieder verwendbares **Binding**. Wird eine Netzwerkadresse mit solch einem Binding assoziiert definiert dies einen **Port** und somit einen einfachen Kommunikationsendpunkt. Ein **Service** ist dann eine Menge zusammengehöriger Ports.

Ein Beispieldokument könnte in etwa folgendermaßen aussehen:

```
<wsdl:definitions name= 'BspImpl'
      targetNamespace= 'http://www.wsdlbeispiel.de/wsdl/BspImpl/' ...>

      <wsdl:types> ... </wsdl:types>

      <wsdl:message name= 'MsgIn'>
            <wsdl:part name= 'name' type='xsd:string' />
      </wsdl:message>

      <wsdl:message name='MsgOut'>
```

[14] iX /Web-Programmierung/
[15] Vgl. W3C /WSDL/

```
            <wsdl:part name= 'Result' type= 'xsd:string' />
      </wsdl:message>

      <wsdl:portType name='BspType'>
            <wsdl:operation name= 'BspOp'>
                  <wsdl:input name='BspIn' message= 'tns:MsgIn'/>
                  <wsdl:output name='BspOut' message= 'tns:MsgOut'/>
            </wsdl: operation>
      </wsdl:portType>
      <wsdl:binding name= 'BspBinding' type= 'tns:BspType'>
            <wsdl:operation name= 'BspOp'>
                  <soap:operation soapAction=
                  'http://www.wsdlbeispiel.de/BspOp' style= 'rpc' />
                  <wsdl:input name= 'TestIn'></wsdl:input>
                  <wsdl:output name= 'TestOut'></wsdl:output>
            </wsdl:operation>
      </wsdl:binding>


      <wsdl:service name='BspService'>
            <wsdl:port name='BspPort' binding='tns:BspBinding'>
                  <soap:address location='http://www.wsdlbeispiel.de/bsp'/>
            </wsdl:port>
      </wsdl:service>

</wsdl:definitions>
```

DISCO

DISCO bietet einen Weg, Beschreibungen von Diensten auf entfernten Rechnern zu finden und zu erhalten. Durch das Discovery Document Format (im XML-Format) kann man ein Discovery-Dokument an einen entfernten Server senden und die WSDL-Beschreibung eines Dienstes empfangen, falls er existiert.

Eine XML-basierte Datei mit der Endung .disco wird in der virtual root eines Servers gespeichert. Diese Dateien linken zur eigentlichen Beschreibung der Dienste, wie auch auf andere .disco-Dateien. Somit umfassen sie die Beschreibungen von Services auf einem oder mehreren Hosts.

Auch kann eine .vdisco Datei anstelle der .disco abgelegt werden. Dann sucht der Discovery Request Handler nach allen Dateien mit Web Service-Endung in jeder virtual root und erstellt dann eine .disco Datei zur Verlinkung auf die Web Service Beschreibung.

UDDI[16]

Wurde ein Dienst erstellt soll dieser im Normalfall auch potentiellen Nutzern zur Verfügung gestellt werden. Zum einen bleibt dem Provider die Möglichkeit, diesen selbst zu propagieren.

[16] Vgl. UDDI.org /Technical White Paper/

Dies kann z.B. ein Hinweis auf einer Webseite erledigen, welcher jedoch nur gefunden werden kann, wenn man auf die Seite aufmerksam gemacht wurde. Zudem kann er so nur von Personen gefunden werden, Anwendungen sind davon ausgeschlossen. Weitreichendere Möglichkeiten bietet da die Veröffentlichung in einem Verzeichnis von Web Services. Zu diesem Zweck wurde UDDI (Universal Description, Discovery and Integration) konzipiert.

UDDI stellt eine Art „Gelbe Seiten“ für WebServices dar. Es handelt sich um verschiedene Verzeichnisse und Bibliotheken, welche Dienstbeschreibungen im XML-Format enthalten. Sie dienen damit der Veröffentlichung und dem Auffinden von gewünschten Web Services. Für das Auffinden stellen sie dabei verschiedene Suchmöglichkeiten zur Verfügung, anhand derer man nach Kriterien wie Unternehmen oder Dienstfunktionen suchen kann.
Der Vorteil von UDDI: nicht nur Personen können nach gewünschten Web Services suchen (was auch über Google o.ä. Suchmaschinen möglich wäre), sondern auch Anwendungen. Der Zugriff ist somit über Frontends in Form von Web-Seiten, als auch über bestimmte API-Funktionen programmatisch möglich.

UDDI umfasst sowohl eine Spezifikation für verteilte, WEB-basierte Informationsregistrierungen für Web Services als auch eine Reihe öffentlich zugänglicher Implementierungen dieser Spezifikation.
In den Registern werden also Informationen über Web Services abgelegt. Zu jedem Dienst gibt es eine Reihe von XML-Datein, die ein Unternehmen und die von ihm angebotenen Dienste beschreiben. Stellt eine Applikation eine Anfrage an ein UDDI-Register, so liefert dieses je nach Bedarf Informationen über neue Dienste, Status von Services sowie über die Verfügbarkeit von kompatiblen Diensten.

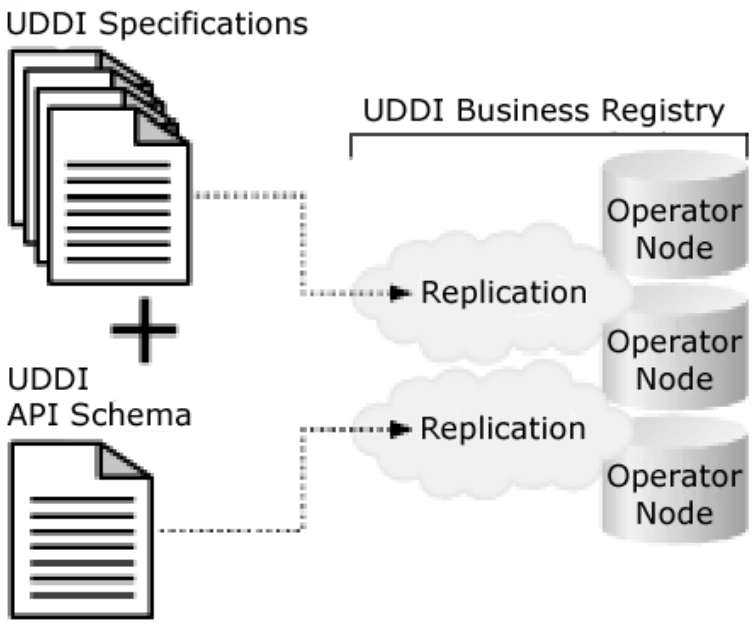
Das Informationsmodell

Konzeptuell kann man 3 Arten von Informationen in eine UDDI Registry eintragen. Sie gehören nicht direkt zur Spezifikation, geben aber einen guten Überblick darüber, was UDDI Verzeichnisse für Daten enthalten:
Die „**White Pages**“ beschreiben das anbietende Unternehmen, mit Informationen wie Name, Adresse usw.
Die „**Yellow Pages**“ beinhalten verschiedene Industrie-Kategorien, zusammengestellt z.B. nach der Standard Industrial Classification. Somit lässt sich ein Unternehmen nach Kriterien wie Produktpalette und geografischer Lage klassifizieren.

Die „**Green Pages**“ beschreiben dann den eigentlichen Dienst. Neben allgemeinen Beschreibungen gehört dazu insbesondere die Beschreibung des Interface des Services, detailliert genug, damit eine Anwendung diesen nutzen kann. Dies beinhaltet Referenzen auf Web Service Spezifikationen, genauso wie Pointer auf Datei- und URL-gestützte Discovery-Mechanismen.

Der Aufbau des UDDI Projektes

Zu UDDI gehören, wie eingangs schon erwähnt, sowohl Spezifikationen als auch eine Reihe von Implementierungen dieser Spezifikationen.
Der Zusammenhang ist in folgender Grafik gut verdeutlicht:



Der Aufbau von UDDI [O'Reilly /UDDI/]

Die Spezifikation

Die UDDI Spezifikation enthält die eigentlichen Informationen über Aufbau und Funktionsweise einer UDDI Registry. Zum einen gibt es Spezifikationen für die Erstellung von Operator Nodes, d.h. Knoten, die den UDDI Mechanismus implementieren. Diese umfassen Dokumente zur Beschreibung des Replikationsprozesses zwischen den Knoten und Informationen darüber, wie sich ein Operator Node verhalten muss. Dazu gehören die Unterstützung von Funktionen für Veröffentlichung und Abfrage von Informationen, aber auch Informationen über die korrekte Datenhaltung.

Daneben enthält UDDI noch die Programmer's API, welche die Funktionen zur Abfrage und Veröffentlichung beschreibt, sowie die dafür benötigten Datenstrukturen. Die Funktionen selber werden durch bestimmte SOAP-Nachrichten ausgeführt, und enthalten zur Datenübermittlung verschiedene XML-Strukturen mit den entsprechenden Daten. Zur eindeutigen Definition sowohl der Strukturen als auch der zugehörigen Typen für diese Nachrichten gibt es noch das UDDI API XML Schema.

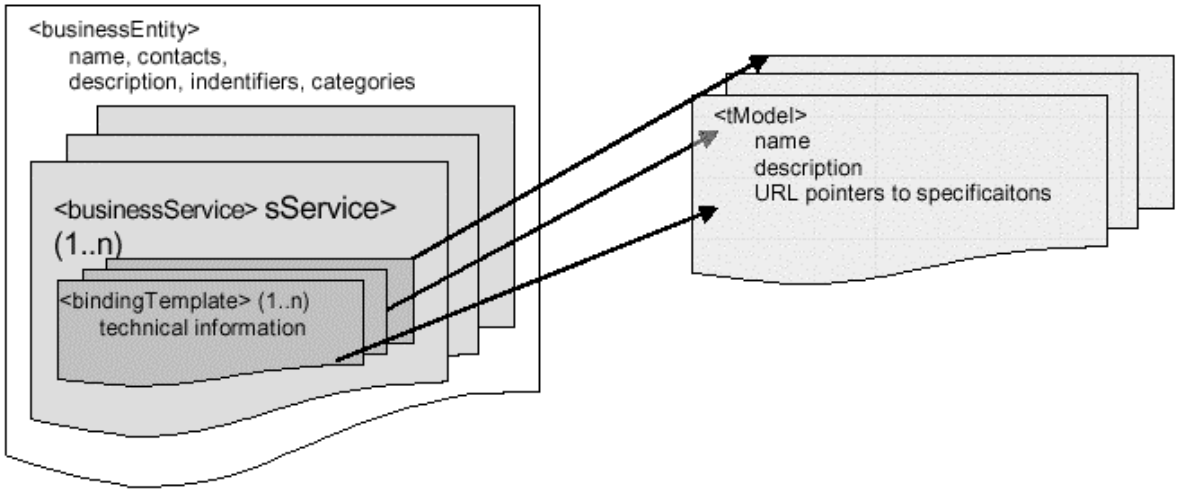
Das XML Schema, das Informationsmodell, definiert 4 Informationstypen, die man braucht, um eine Web Service zu beschreiben und zu nutzen. Dies sind die business information, service information, binding information, und Informationen über Servicespezifikationen, die technical information. Die Informationen werden in Form zugehöriger Datenstrukturen in den API-Aufrufen als In- und Outputparameter verwendet.

Die **business entity** beschreibt den Anbieter eines Dienstes, welcher zumeist ein Unternehmen darstellen wird. Dazu gehören Angaben wie Name, Kontaktinformationen oder Kategorisierung.

Eine business entity enthält mehrere **business service** Elemente. Dieses beschreibt eine Reihe zusammengehöriger Dienste, die vom Unternehmen angeboten werden. Das Element ist wieder verwendbar und kann damit für mehrere business entities eingesetzt werden, z.B. wenn Subunternehmen selbst im Verzeichnis vorhanden sind aber denselben Dienst einsetzen.

Binding Templates sind Bestandteile von business services und enthalten grundlegende Informationen eines Dienstes. Dazu gehören Verweise auf technische Beschreibungen und die Zugriffsadresse des Dienstes, jedoch keine Details über dessen Implementierung. Sie enthalten somit die Informationen, die zum Benutzen eines Web Service nötig sind.

Auch gehört zu einem Binding Template eine Referenz auf ein oder mehrere **tModel** Elemente. Es ist die abstrakte Beschreibung der Spezifikationen und des Verhaltens eines Web Services. Es kann als „fingerprint" dafür angesehen werden, wie man mit einem Dienst interagieren kann. Jedoch enthält das tModel die Spezifikationen des Dienstes nicht direkt, sondern stellt Verweise auf entsprechende Dokumente zur Verfügung, wie z.B. auf ein WSDL Dokument. Durch diese Informationen kann ein Nutzer feststellen, ob der Dienst kompatibel zu seinen Anforderungen ist.



Die UDDI XML Elemente [UDDI.org /Technical White Paper/]

Für den Zugriff auf die UDDI Registry über Programme gibt es eine entsprechende API. Sie wird eingeteilt in Funktionen für die Abfrage (Inquiry) und Veröffentlichung (Publish).

Die Implementierung

Die umfassendste öffentliche Implementierung der UDDI-Spezifikationen ist die UDDI Business Registry (UBR), auch die „Public Cloud“ genannt. Sie ist eine öffentliche Implementierung der UDDI-Spezifikation. Konzeptuell gesehen ist dies ein einzelnes System, welches auf vielen Knoten, aufgebaut ist. Die Knoten werden von namhaften Unternehmen wie IBM oder Microsoft gehostet. Diese Knoten synchronisieren ihre Daten durch Replikation. So halten alle Knoten die gleichen Daten. Die UBR ist dann eine globale Gruppierung von Operator Nodes.
Ein eigener Operator Node muss aber nicht Bestandteil der UBR sein. Unternehmen können eigene private Operator Nodes installieren. Mehrere private Knoten können auch hier zusammengefasst werden und ihre Daten replizieren, ohne mit der UBR zu synchronisieren. Dies bildet dann eine „Private Cloud“. So lassen sich firmeninterne Dienste im Intranet zur Verfügung stellen, oder man kann Dienste einer anderen Gruppe von Nutzern zur Verfügung stellen.

Plattformen

Die Entwicklung eines Web Services kann insbesondere durch die recht umfangreiche Beschreibung in WSDL ziemlich aufwendig werden – ein Grund, sich nach einer geeigneten Entwicklungsumgebung umzusehen. Auch zur Erstellung einer Anwendung, die einen Web Service nutzt, kann sich durch die Umsetzung der Dienst-WSDL recht umfangreich gestalten. Somit kann hier ebenso auf eine Entwicklungsumgebung zurückgegriffen werden. Und hat man einen Dienst entwickelt, benötigt man noch eine Umgebung, auf der er arbeitet und seine Funktionen erfüllt.
Um den Entwicklungsprozess zu erleichtern und eine Laufumgebung für Web Service zu haben eignet sich hervorragend der Einsatz entsprechender Plattformen.

Bei diesen lassen sich grundsätzlich zwei „Lager“ ausmachen. Auf der eine Seite Microsoft mit seiner .NET Umgebung, auf der anderen die J2EE Technologie[17].
Beiden Plattformen zugleich ist, dass sie die genannten Standard-Technologien wie WSDL oder SOAP unterstützen. Somit können sie entsprechend den geforderten Standards und Eigenschaften Web Services entwickeln, die nicht nur mit Web Services der gleichen Plattform, aber auch mit denen von komplett anderer Systemen interoperabel sind.

Microsoft .NET

Als Microsoft-Produkt ergibt sich der erste Fakt sofort: .NET ist nur für den Einsatz auf Windows-Betriebssystemen konzipiert. Somit müssen sich Nutzer anderer Betriebssysteme nach anderen Anwendungen umsehen. Dafür gibt es aber einen Vorteil zum Ausgleich: bei der Entwicklung mit .NET ist man nicht auf die Nutzung einer bestimmten Programmiersprache beschränkt. Zurzeit gibt es Compiler für mehr als 15 Sprachen, wie C++, C# oder Visual Basic. Zudem bietet die Entwicklungsumgebung Visual Studio .NET umfangreiche Tools, die zur Entwicklung von Web Services herangezogen werden können.

J2EE

J2EE ist eine Standardtechnologie für die Entwicklung von Multi-Tier-Unternehmensanwendungen. Es baut auf der Java 2 Standardedition auf, bietet aber zusätzliche API’s für die Entwicklung im Unternehmensbereich. Ursprünglich für den Aufbau

[17] CC eGov /Web Services/

von Anwendungen, Softwarekomponenten und herkömmlichen Webseiten entwickelt, lassen sich durch Erweiterungen um Web Service API's auch auf der J2EE Plattform Web Services entwickelt.
Es existieren diverse Unternehmen, die eigene Umsetzungen der J2EE anbieten, wie IBM oder SUN. Man kann sich somit für ein System entscheiden, welches den eigenen Bedürfnissen entspricht. Allen zugleich ist, dass als Programmiersprache Java verwendet werden muss – ein Manko für Entwickler, die noch keine Java-Erfahrungen gesammelt haben. Dafür sind die Plattformen auf nahezu allen existierenden Plattformen einsetzbar.

Ob .NET oder J2EE – beide Konzepte bieten Vorteile auch nach der Erstellung eines Dienstes. Durch das Ausführen der Anwendungen in Containern können über diese zusätzliche Funktionen realisiert werden. Dazu gehören unter anderem die Sicherheit der Datenübertragung oder die Gewährleistung bestimmter Performance-Eigenschaften – entscheidende Punkte für den erfolgreichen Einsatz der Technologie im Unternehmensbereich.

Anforderungen an Web Services

Sicherheit

Damit Web Services erfolgreich werden können müssen sie dem Anspruch an Sicherheit genügen. Nur so können sie erfolgreich im Bereich des e-Business eingesetzt werden.
Zu den Anforderungen bezüglich der Sicherheit gehören:

- **Integrität** – wurde die Nachricht auch nicht verändert, sei es zufällig oder bewusst?
- **Vertraulichkeit** – kein Zugang für unautorisierte Personen, Prozesse, …
- Identität des Senders/ **Beweis der Herkunft** – Beweis der Herkunft der Nachricht/Daten; übertragen durch identifizierten Sender
- **Autorisierung** – Gewährung von Zugriffsrechten, Garantie das der Sender autorisiert ist, die von der Nachricht geforderte Operation auszuführen

Es gibt verschiedene Ansätze, die diese Anforderungen erfüllen sollen. Dabei scheint eine Technik nicht auszureichen, um alle Gebiete abzudecken. Im folgenden Teil gebe ich einen

kurzen Überblick über vorhandene Techniken[18]. Diese gehen über SSL, welches zur Sicherung von Ende-zu-Ende Verbindungen eingesetzt wird, hinaus.

XML-Encryption / XML-Signature

Komplette XML-Dokumente können durch SSL sicher an Empfänger gesendet werden. Jedoch lässt dies keine Möglichkeit, Berechtigungen auf bestimmte Teile eines Dokumentes festzulegen. Dies ist oftmals sinnvoll, wenn die Daten im Unternehmen von verschiedenen Instanzen ausgewertet werden, die alle nur einen Teil des Dokumentes heranziehen sollen. XML-Encryption und XML-Signature sind Verfahren, welche die beschriebene Funktionalität zur Verfügung stellen können.

XML Key Management Specification (XKMS)

XKMS ist eine Spezifikation für sichere Web Services. Diese ermöglicht Web Services, kryptographische Schlüssel, welche z.B. für digitale Unterschriften und Verschlüsselung eingesetzt werden, zu registrieren und zu verwalten. Dadurch kann Authentisierung gesichert werden.

WS-Security

WS-Security ist eine Spezifikation für die Sicherheit von Web Services. Erarbeitet wurde diese in einer Zusammenarbeit von IBM, Microsoft und VeriSign. Es handelt sich hierbei um eine Erweiterung von SOAP. Es werden viele Sicherheitsmodelle und Verschlüsselungstechnologien (wie auch XML Encryption) unterstützt. Ziel ist, für die Sicherheit und Integrität der Web Service Daten zu sorgen.

SAML

SAML ermöglicht die Authentifikation. Dadurch kenn der Nutzer mehrere Web Services ohne jeweilige Neuanmeldung nutzen. Zurzeit handelt es sich bei SAML noch um keinen akzeptierten Standard.

SOAP-DSIG

SOAP-DSIG definiert Syntax und Regeln für digital signierte SOAP-Nachrichten. Es wird eingesetzt für die Authentifizierung von Nachrichten. Hierfür wird durch ein neues Datenformat eine XML-Signatur an eine SOAP-Nachricht angehängt.

[18] Vgl. CC eGov /Web Services/

Damit auch die Authentizität des Senders gesichert wird ist eine Kombination mit z.B. SSL nötig.

HTTPR

HTTPR definiert die Art, wie Metadaten und Applikationsnachrichten im HTTP Protokoll eingekapselt werden. Es erlaubt damit sicherzustellen, dass eine Nachricht dem richtigen Empfänger genau einmal übermittelt wurde. Dafür führt man sechs neue Übermittlungsvereinbarungen ein, welche als „command flow" bezeichnet werden.

Transaktionsfähigkeit

Transaktionssicherheit ist eine häufige Anforderung an verteilte Systeme, besonders im Unternehmensbereich. Verwendet man Web Services muss bei komplexen Geschäftsprozessen, die sich über mehrere Funktionen und längere Zeiträume erstrecken, mit der Nichtdurchführbarkeit bestimmter Operationen gerechnet werden. Solch ein Fehlschlag kann sowohl durch nicht erfüllte Geschäftsbedingungen als auch durch technische Fehler auftreten. Genau hierfür wird ein Transaktionssystem benötigt, welches auch nach Fehlern einen konsistenten Zustand hinterlässt.
Dazu wird bei Transaktionen eine Folge von Operationen zu einer gemeinsam zu verarbeitenden Einheit zusammengefasst. Somit soll die Integrität betrieblicher Daten gesichert werden. Schlägt eine der Operationen einer Transaktion fehl, werden die schon durchgeführten Aktionen wieder rückgängig gemacht, um den konsistenten Zustand wieder herzustellen.

Die erklärten Standards, die den Web Services zugrunde liegen, bieten alleine keine Vorkehrungen für die Gewährleistung der Transaktionssicherheit. Somit werden eine Reihe zusätzlicher Techniken benötigt. Auf diese werde ich im Rahmen dieser Arbeit aber nicht eingehen.[19]

Performance

Um erfolgreich zu sein sollten heutige Kommunikationsmodelle ansprechende Geschwindigkeit bieten können. Das Problem der Web Services: die verwendeten Protokolle

[19] ausführliche Informationen unter Geiß /transaktionssichere Web-Service-Anwendungen/

sind ziemlich „fett“. Die Interaktionen über das Netzwerk sind relativ langsam und verbrauchen viel Bandbreite. Der Grund: SOAP ist XML-basiert. Jedes Datenfeld besitzt ein text-basiertes Tag, und Daten werden im Normalfall ebenso als normaler ASCII-Text übertragen. Dadurch wird das Datenvolumen um Einiges erhöht.
Eine Möglichkeit zur Kompensation ist der Ausbau der Infrastruktur. Dies ist jedoch oftmals nicht im Bereich des Möglichen. Eine andere „Lösung“, das Wählen sehr kurz gehaltener Namensraumbezeichnungen, schränkt wiederum die Verständlichkeit stark ein.
Nähere Informationen zur Performance finden sich in einem der weiteren Vorträge zum Thema.

Quality of Service

Gerade im Unternehmensbereich spielt die Qualität der verwendeten Software eine entscheidende Rolle. Ein Unternehmen, welches einen Dienst in Anspruch nimmt, erwartet auch ein bestimmtes Maß an Qualität. Der Dienst sollte die an ihn gestellten Leistungen so erbringen wie das erwartet wird.
Jedoch gerade in einem Szenario, wo Dienste erst zur Laufzeit eingebunden werden und man somit keinen direkten Bezug zum Unternehmen hat fällt es schwer, die Qualität des Dienstes zu erkennen.
Für dieses Problem scheint es im Moment noch keine geeigneten Lösungen zu geben. Denkbar ist z.B. eine entsprechende Bewertung in den UDDI Verzeichnissen. Konkrete Umsetzungen dazu gibt es aber anscheinend nicht.

Verfügbarkeit

Ebenso wichtig wie die Qualität einer erbrachten Leistung ist auch die Verfügbarkeit. Oftmals ist es sehr wichtig, dass man stets einen bestimmten Dienst zur Verfügung hat. Ist dies nicht möglich, können ganze Geschäftsprozesse ins Stocken kommen. Jedoch besteht in einer verteilten Umgebung immer die Gefahr, dass Teile eines Systems nicht erreichbar sind. Gründe dafür gibt es genug – Ausfall des Netzes oder Probleme mit dem hostenden Rechner sind nur zwei Beispiele.
Bei vollständiger Umsetzung des Konzeptes der Web Services sollten diese in der Lage, die Verfügbarkeit nahezu immer zu gewährleisten. Fällt ein ausgewählter Dienst aus, kann man sich über eine Suche in einem UDDI Verzeichnis einen gleichwertigen Dienst suchen, der die

benötigten Funktionen zur Verfügung stellt. Zwar wird es nicht immer einen passenden Ersatz geben, aber für viele Dinge gibt es sicher mehrere Implementierungen.
Geht das eben genannte Konzept nicht auf oder liegt das Problem im eigenen Unternehmen, ist die Verfügbarkeit jedoch genauso ein Problem wie bei anderen Techniken im Bereich verteilter Systeme.

Was bringen uns Web Services?

Mit Web Services sind viele Erwartungen verbunden. Am Anfang dieser Abfassung wurden schon Szenarien für den Einsatz von Web Services in der Unternehmenswelt angesprochen. Nun wird noch etwas detaillierter auf den erwarteten Nutzen eingegangen. Der Aspekt der Beziehung Unternehmen - Kunde wird dabei nicht betrachtet.

Sicher verbinden unterschiedliche Unternehmen auch unterschiedliche Erwartungen an Sinn und Nutzen von Web Services. Während einige sich viele positive Auswirkungen erwarten, sehen andere die Technik mit mehr Abstand. Gerade in der Zeit, die bis zur Etablierung der Web Services vergeht, sehen einige Unternehmer noch ein schwer kalkulierbares Risiko in Investitionen in die Entwicklung von Web Services. Allgemein ist jedoch eine positive Erwartungshaltung auszumachen. Auf einige dieser Aspekte möchte ich kurz eingehen[20]. Dabei ist sicher erwähnenswert, dass bestimmte Punkte auch durch andere Techniken als Web Services erfüllt werden können. Trotzdem gehören sie zum an diese gestellten Erwartungsbild dazu.

Vorhandene Investitionen können genutzt werden.
Die Neuentwicklung von Unternehmensanwendungen kostet zumeist viel Zeit und Geld. Web Services bieten die Möglichkeit, bereits vorhandene Systeme auch dann beizubehalten, wenn erweiterte Funktionalitäten gefragt sind. Oder neue Funktionen überhaupt erst anzubieten. Neue Nutzungsmöglichkeiten werden durch Hinzufügen von Funktionalität durch Web Services aufgeschlossen.

[20] Attachmate /Nutzen von Web Services/
Conner /WS: Next Horizon/

So kann mitunter auf komplette Neuentwicklungen verzichtet werden, oder bisher wegen Kostenaufwand aufgeschobene Entwicklungen können durchgesetzt werden. Bestehende Legacy-Systeme lassen sich in neue e-Business-Anwendungen integrieren. Neue Dienste können so kostengünstig und mit relativ wenig Zeitaufwand Partnern und Kunden angeboten werden.

Neue Anwendungen lassen sich schnell entwickeln.

Web Services lassen sich schnell mit z.T. bereits bekannten Tools entwickeln. Eine aufwendige Umarbeitung ist oftmals nicht vonnöten. Tools können die Entwicklung so weit automatisieren, dass aus der Angabe von Schnittstellen und Funktionen der Dienst automatisch erzeugt wird. Somit wird auch die Barriere für den Einstieg in die Entwicklung von Web Services gesenkt.

Komponenten können mehrfach verwendet werden.

Durch Einhaltung der Web Service Standards können schon entwickelte Web Services problemlos in weiteren oder neuen Anwendungen wieder verwendet werden. Man kann den Web Service für viele Zwecke in- und außerhalb eines Unternehmens einsetzen und mit anderen Web Services kombinieren. Plattform-Abhängigkeiten oder Hersteller-Spezifika spielen dabei keine Rolle mehr.

Inhärente Sicherheit

Im Unternehmenssystem kann vom Betreiber entschieden werden, welche Funktionen für andere Nutzer oder Anwendungen verfügbar gemacht werden. Damit kann die Sicherheit im Unternehmen unterstützt werden. Durch die Trennung von Implementierung und Schnittstelle hat der Nutzer keine Möglichkeit, an das unterliegende System zu gelangen. Sein Einblick und Zugriff bleibt auf die Schnittstelle begrenzt.

Neue Möglichkeiten in der Kommunikation/Zusammenarbeit.

Web Services können potentiellen Partnern eine Möglichkeit zur Kommunikation auf Anwendungsebene bieten. Unternehmen, deren Zusammenarbeit aufgrund inkompatibler Systeme scheiterte, haben durch den Einsatz der Web Service Technologie neue Möglichkeiten der Integration ihrer Systeme. Es können neue Integrationen entstehen und Kunden hinzugewonnen werden.

Einsparungen durch Web Services.
Managementkosten und Infrastruktureinsatz werden vergünstigt. Widerverwendbarkeit spart Kosten für Neuentwicklungen.

Beispiele für den Einsatz

Neben der Theorie möchte ich noch ein paar Beispiele für den Einsatz für Web Services vorstellen. Diese können sowohl einfache Aufgaben, wie das einfache Liefern eines Wertes (Uhrzeit) übernehmen, aber auch für komplexe Geschäftsprozesse eingesetzt werden.

Kalender als Web Service
Im Bereich der Privatanwender könnte sich ein Terminkalender als Web Service realisieren lassen. Neben der einfachen Pflege seitens des Besitzers stünden erweiterte Verwaltungsmöglichkeiten zur Verfügung. So könnten z.B. Werkstätten oder Ärzte ihre eigenen Kalender mit dem des Nutzers verbinden und Termine mit dessen Kalender synchronisieren.

Mitarbeiter-Informationssystem.
Oftmals sind Informationen über Mitarbeiter auf viele Systeme verteilt. Gehaltsdaten, Sozialsysteme usw. sind nicht integriert.
Ein Ausweg wäre eine Internet-Anwendung, welche den Mitarbeitern einen Zugang zu ihren Daten gewähren könnte. Der Telefonverkehr zur Personalabteilung und somit das dortige Personal wären entlastet.
Schrittweise könnte das System auch für den Zugang von außen vorbereitet werde, so dass z.B. Sozialversicherungsträger Zugang zu mitarbeiterrelevanten Informationen erhalten könnten.

Preis- und Lagerdaten für Partner.
Für Partner, die qualifiziert genug sind, können Web Services Einblicke in den eigenen Lagerbestand und Preislisten gewähren. Die Entwickler des Partners können die Web Services dann für eigene e-Business-Anwendungen einsetzen.

Einschätzung

Viel wurde über Web Services gesprochen und geschrieben. Nach dem anfänglichen Hype, verbunden mit vielfältigen Vorschusslorbeeren und z.T. übertriebenen Erwartungen kehrt in der Medienwelt Ruhe ein. Die Realität hat die Analysten eingeholt, und es werden realistisch Vor- und Nachteile der neuen Technologie abgewogen. Dementsprechend möchte ich eine kurze Zusammenfassung geben.
Kommen wir zuerst zu (teils schon erwähnten) Nachteilen.

Das große Ziel der uneingeschränkten Interoperabilität ist heute sicher noch nicht erreicht[21]. Speziell im Bereich der SOAP-Kommunikation gibt es teils Probleme durch unterschiedliche Implementierung der SOAP-Spezifika. So gibt es z.B. bei SOAP::Lite und .NET Unterschiede bei der Generierung von SOAP Action Headern oder der Art der Parameterbenennung, was schließlich zu Inkompatibilitäten führen kann[22]. Es kann nicht garantiert werden, das mit daraus resultierenden Anpassungen, welche die Funktion ermöglichen, Probleme mit anderen Systemen auftreten. Hier herrscht sicher noch Handlungsbedarf.
Des Weiteren kann die Unabhängigkeit von Plattformen und Protokollen, d.h. von engen Übereinkünften im Allgemeinen, nicht überall ihre Stärken ausspielen. Gerade in komplexeren Unternehmensanwendungen, seien dies Bankensysteme oder Versicherungsanwendungen, sind auf höherem Level auch weiterhin enge Übereinkünfte nötig. Nur so können spezialisierte Funktionen realisiert werden.
Neben den technischen Einschränkungen gibt es noch eine weitere Frage. Diese behandelt das Thema, ob und in welchem Maße die Web Service Technologie Verwendung finden wird. Unternehmen müssen sich fragen, ob die Investition in die Entwicklung von Diensten sowohl wirtschaftlich als auch technisch lohnenswert ist. Trotz positiver Erwartungen ist ein breiter Erfolg noch nicht garantiert.
Und der Bereich der bezahlten anonymisierten Dienste, die ohne direkten (persönlichen) Kontakt mit Unternehmen verwendet werden, wird sicher nicht überall auf Zuspruch treffen. Lässt sich doch schlecht erklären, warum der Kunde den eigenen teureren Dienst nutzen soll, wenn bei der Konkurrenz günstigere zur Verfügung stehen. Dafür bedarf es persönlicher Gespräche oder anderen Kontaktes, um bestimmte Vorzüge zu erläutern.[23]

---

[21] xMethods /SOAP Interoperability/
[22] XML.com /WS Interoperability/
[23] Starke /Web Service Engineering/

Jedoch scheint sich abzuzeichnen, dass sich Web Services in Zukunft durchsetzen werden können.

Dafür spricht mit Sicherheit die Architektur der Web Services. Sie ermöglicht relativ viele Freiheiten in der Entwicklung (frei wählbar sind Programmiersprache und Plattform). Wenn richtig umgesetzt entfallen durch die Verwendung von Standards Probleme durch Inkompatibilitäten zwischen herstellerfremden Systemen. Und am wichtigsten: Web Services können auf bereits bestehenden Technologien und Infrastruktur aufgesetzt werden. Dies ist eine gesicherte Grundlage für einen weitreichenden Erfolg der Technologie.

Probleme wie der Overhead bei der Datenübertragung, basierend auf der Verwendung von XML, lassen sich durch Mittel wie Komprimierungsverfahren sicher relativieren, oder machen im Einsatzgebiet nicht so viel aus. Aspekte, die für den Einsatz im Unternehmensbereich von Bedeutung sind, wie Sicherheit und Transaktionssteuerung, finden in Form verschiedener Implementierungen schon eine Umsetzung.

Somit können Web Services wohl einer recht erfolgreichen Zukunft entgegen sehen. Durch ihre Flexibilität, die Möglichkeit des dynamischen Aufsuchens und Bindens, kombiniert mit den weiteren positiven Eigenschaften, sind sie für zukünftige Bedürfnisse entsprechend vorbereitet.

Literatur und Links

W3C /Web Service Architecture/

Web Service Architecture, 06/2002

http://www.w3.org/2002/ws/arch/2/06/wd-wsa-arch-20020605.html

W3C /WSDL/

Web Services Description Language (WSDL) 1.1, 05/2001

http://www.w3.org/TR/wsdl

W3C /WS Architecture Description/

Web Service Architecture – Description, 10/2002

http://lists.w3.org/Archives/Public/www-ws-arch/2002Oct/att-0406/01-HKsContribution.description.htm

iX /Web-Programmierung/

Lars Röwekamp, iX 9-11/2002: Web-Programmierung

http://www.heise.de/ix/artikel/2002/09/121/

Gottschalk /WS Architecture/

K. Gottschalk, S. Graham, H. Kreger, J.Snell, 11/2001

http://www.research.ibm.com/journal/sj/412/gottschalk.html

Lau /Developing Web Services/

C. Lau, A. Ryman: Developing XML Web services with WebSphere Studio Application Developer, 01/2002

http://www.research.ibm.com/journal/sj/412/lau.html

Stuart /XML Schema/

Ian Stuart: XML Schema, a brief introduction

http://lucas.ucs.ed.ac.uk/xml-schema/

Horn /SOAP, WSDL, UDDI/

Thorsten Horn: Web Services mit SOAP, WSDL und UDDI

http://www.torsten-horn.de/techdocs/soap.htm

Gisolfi /dynamic e-business/

Dan Gisolfi: An introduction to dynamic e-business, 04/2001

http://www-106.ibm.com/developerworks/webservices/library/ws-arc1/

UDDI.org /Technical White Paper/

UDDI Technical White Paper, 06/2000

http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf

O'Reilly /UDDI/

Tyle Jewell, David Chappell : Java Web Services, 05/2002

Chapter 6: UDDI

http://www.perfectxml.com/oreilly/chapter.asp?row_id=18

CC eGov /Web Services/

Web Services

http://webservice.iwv.ch/

xMethods /SOAP Interoperability/

SOAPBuilders Interoperability Lab

http://www.xmethods.net/ilab/

XML.com /WS Interoperability/

James Snell: Web Services Interoperability, 01/2002

http://www.xml.com/pub/a/2002/01/30/soap.html

Webservices.XML.com /SOAP/

Rael Dornfest, Clay Shirky: Emerging Technology Briefs: SOAP, 04/2002

http://webservices.xml.com/pub/a/ws/2002/04/16/soap.html

Geiß /transaktionssichere Web-Service-Anwendungen/

Matthias Geiß: Architektur für transaktionssichere Web-Service-Anwendungen, 12/2002

http://www.forsoft.de/zen/teaching/DA_Geiss.pdf

Starke /Web Service Engineering/

Gernot Starke: Web Service Engineering,

http://www.sigs.de/publications/os/2002/01/starke_OS_01_02.pdf

Glass /WS (r)evolution/

Graham Glass: The Web Service (r)esolution; Applying Web services to applications, 11/2000

http://www-106.ibm.com/developerworks/library/ws-peer1.html

Attachmate /Nutzen von Web Services/

Attachmate Corporation: Der Nutzen von Web Services, 05/2002

http://de.attachmate.com/article/0,1012,3856_14,00.html

Conner /WS: Next Horizon/

Mike Conner: Web Services: The Next Horizon for e-business

ftp://ftp.software.ibm.com/software/websphere/webservices/ws-next-horizon.pdf