Georg-August-Universität Göttingen

Institut für Wirtschaftsinformatik
Professor Dr. Matthias Schumann

Platz der Göttinger Sieben 5
37073 Göttingen
Telefon: + 49 551 39 - 44 33
+ 49 551 39 - 44 42
Telefax: + 49 551 39 - 97 35
www.wi2.wiso.uni-goettingen.de

Arbeitsbericht Nr. 22/2003

Hrsg.: Matthias Schumann

Markus Burghardt / Svenja Hagenhoff

Web Services – Grundlagen und Kerntechnologien

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ADS	Advertisement and Discovery of Services
API	Application Programming Interface
AXIS	Apaches Extensible Interaction System
CDR	Common Data Representation
CORBA	Common Object Request Broker Architecture
DAML-S	DARPA Agent Markup Language Ontology for Web Services
DCOM	Distributed Common Object Model
DIME	Direct Internet Message Encapsulation
DTD	Document Type Definition
DOM	Document Object Model
GUID	Globally Unique Identifier
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-ORB-Protocol
IOR	Interoperable Object Reference
NDR	Network Data Representation
RDF	Resource Description Framework
RMI	Remote Method Invocation
RPC	Remote Procedure Call
OMA	Object Management Architecture
OMG	Object Management Group
OQL	Object Query Language
ORB	Object Request Broker
SAX	Simple API for XML
SGML	Standard Generalized Markup Language

SQL	Structured Query Language
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
URI	Uniform Ressource Identifier
URL	Uniform Ressouce Locator
WDDX	Web Distributed Data Exchange
WSDL	Web Service Description Language
WSTK	Web Services Toolkit
XML	Extensible Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations
XSL-FO	XSL Formatting Objects
W3C	World Wide Web Consortium

1 Einleitung

Web Services ermöglichen die Realisierung verteilter Anwendungen. Dabei werden sie nicht als eine neue Technologie aufgefasst, sondern können als eine Kombination bereits etablierter Technologien angesehen werden, die im Zusammenspiel plattform- und programmiersprachenunabhängige verteilte Anwendungen ermöglichen.[1] Web Services vereinfachen somit die Kommunikation zwischen Anwendungen in heterogenen Netzwerken wie beispielsweise dem Internet. Web Services basieren auf der Metasprache XML sowie standardisierten etablierten Internetprotokollen wie TCP/IP und HTTP.[2]

An der zur Zeit geführten Diskussion im Umfeld des Internets fällt auf, dass Web Services ein neues Schlagwort in diesem Bereich sind, dem eine sehr große Bedeutung zugemessen wird. Dieses wird auch durch zahlreiche auf Web Services spezialisierte Webseiten und Fachzeitschriften deutlich.[3] Darüber hinaus wird diese Beobachtung durch aktuelle Umfragen von IDC und Gartner bestätigt, die in diesem Bereich einen starken Anstieg der Umsätze von derzeit 1,6 Milliarden auf über 30 Milliarden US-Dollar bis in das Jahr 2005 erwarten.[4] Auch große Unternehmen aus dem IT-Bereich wie beispielsweise IBM, Microsoft und SUN fördern Web Services und beteiligen sich aktiv an der Spezifikation der relevanten Technologien. Bei der Suche nach aktuell verfügbaren Web Services bieten u. a. die Seiten von XMethods (www.xmethods.net) und Salcentral (www.salcentral.com) einen gute Auflistung von derzeit ca. 550 aktiven Web Services. Auch die Wissenschaft kann sich diesem neuen Trend nicht entziehen und so sind in der letzten Zeit vermehrt Workshops und Konferenzen zur dieser aktuellen Thematik aufzufinden.[5]

Web Services stehen jedoch nicht in einem direkten Konkurrenzverhältnis zu bereits verbreiteten Technologien für verteilte Anwendungen wie z.B. CORBA oder RMI, denn mit CORBA und RMI ist die Realisierung objektorientierter Anwendungen möglich, wohingegen Web Services vom Grundsatz her nur entfernte Funktionsaufrufe unterstützen.[6] Web Services sollten also in der jetzigen Situation als Ergänzung zu den bereits genannten anderen Technologien gesehen werden.

---

1 Vgl. Knuth (2002), S. 12.
2 Vgl. Burghardt / Gehrke / Schumann (2003a), S. 72f.
3 Vgl. WebServices.Org (2003); o. V. (2003).
4 Vgl. Box / Skonnard / Lam (2000); o. V. (2002b).
5 Vgl. Buchmann / Casati / Fiege / Hsu / Shan (2002); Chaudhri (2003).
6 Vgl. Burghardt / Gehrke / Schumann (2003b), S. 45f.

Ziel dieses Arbeitsberichtes ist es, eine Begriffsdefinition des Schlagwortes Web Service zu erarbeiten und diesem Begriff gegenüber den oben bereits etablierten Technologien abzugrenzen und eine Beurteilung vorzunehmen. Darüber hinaus soll ein grundlegende Architekturmodell darstellt werden. Für die Realisierung der fachlichen Anforderungen innerhalb dieses Architekturmodells haben sich verschiedene Basistechnologien etabliert, die ebenfalls einer Diskussion bedürfen.

Anhand der Zielsetzung ergibt sich der Aufbau des Arbeitsberichts. In einem Grundlagenabschnitt werden eine Begriffsdefinition für Web Services arbeitet und das Architekturkonzept vorgestellt sowie eine Abgrenzung zu verwandten Konzepten vorgenommen. Das Grundlagenkapitel schließt mit einer Herleitung von Schlüsselfaktoren, die eine Eignungsbewertung von Web Services als Schnittstellentechnologie für das Angebot digitaler Dienstleistungen ermöglichen. Im nächsten Kapitel werden die essentiellen Technologien von Web Services diskutiert und eventuelle Alternativen für die einzelnen fachlichen Anforderungen aufzeigt. Der Arbeitsbericht schließt mit einem kurzen Fazit und einem Ausblick auf weitere Forschungsvorhaben.

2 Grundlagen

In diesem Abschnitt werden die Grundlagen der Innovation Web Services dargestellt. Dabei werden ausgehend von einer Begriffsdefinition das grundlegende Architekturmodell vorgestellt und die umgesetzten Kommunikationsparadigmen erläutert. Abschließend findet eine Einordnung in Konzepte zur Realisierung verteilter Anwendungen auf der einen Seite und Middleware auf der anderen Seite statt.

2.1 Begriffsdefinition

Eine Analyse der Literatur und Fachzeitschriften zum Thema Web Services zeigt, dass es keinen Mangel an Definitionen für den Begriff „Web Service“ gibt und nahezu jeder Fachautor seine eigene Begriffsdefinition verwendet. Eine einheitliche, konsistente und standardisierte Begriffsdefinition ist somit zur Zeit nicht existent. Daher sollen nun zwei Definitionen vorgestellt und daraus eine Arbeitsdefinition dieses Begriffs.[7] Dabei fiel die Auswahl auf die Definition des Standardisierungsgremiums W3C, da das W3C als ein wesentlicher Treiber für diese Innovation angesehen werden kann. Eine andere Definition ist aus einem der ersten veröffentlichen Fachbücher zu dieser Thematik von Ethan Cerami entnommen.

Das World Wide Web Consortium (W3C) definiert den Begriff des Web Services wie folgt[8]:

> *“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.”*

Auf der anderen Seite schlägt der Ethan Cerami folgende Definition vor[9]:

> *„A web service is any service that is available over the Internet, uses a standardized XML messaging system, and is not tied to any one operating system or programming language. Although they are not required, a web service may also have two additional properties: A web service should be self-describing and a web service should be discoverable.”*

---

7 Vgl. für einen Überblick von verfügbaren Begriffsdefinitionen Jeckle (2002).
8 Vgl. Austin / Barbir / Ferris / Garg (2002).
9 Vgl. Cerami (2002), S. 3f.

Aus diesen Definitionen können Web Services als Softwarekomponenten oder unabhängige, modulare Dienste aufgefasst werden. Diese Softwarekomponenten kapseln in sich eine wohl definierte Funktionalität und stellen diese über standardisierte Schnittstellen anderen Softwarekomponenten oder Anwendungen zur Verfügung.[10] Durch diese Standardisierung der Schnittstellen ergibt sich die Möglichkeit der losen Kopplung von Web Services untereinander beziehungsweise von Web Services und anderen Anwendungen. Die Nutzer können die Softwarekomponenten ohne Probleme austauschen, wenn sich die Schnittstellensignatur der Dienste nicht unterscheidet. Die Kapselung der Implementierung hinter einer standardisierten Schnittstelle ermöglicht somit eine Umsetzung auf allen Plattformen sowie jeder Programmiersprache (Separation of Concerns).[11] Ein Web Service kann immer über die zugeordneten Verbindungsendpunkte identifiziert werden. Verbindungsendpunkte sind dabei in der Lage, Nachrichten zu empfangen, diese zu verarbeiten und optionale Antwortnachrichten zu erzeugen.[12] Die Verbindungsendpunkte bei Web Services werden mit einem Uniform Resource Identifier (URI) bezeichnet.[13]

Web Services sind darüber hinaus selbstbeschreibend, da die bereitgestellten Funktionalitäten und die zur Nutzung notwendige Schnittstellensignatur in Form von XML-Dokumenten zur Verfügung gestellt werden. Auf Basis dieser Dokumente ist es einem Nutzer möglich, einen Web Service in das entsprechende Anwendungsumfeld einzubinden bzw. zu integrieren.[14] Auch basiert die Suche nach Web Services und letztendlich ihre Nutzung jeweils auf Basisstandards, die auf der Metasprache XML und etablierten Internettechnologien wie beispielsweise TCP/IP und HTTP aufsetzen. Damit übernehmen Web Services Eigenschaften wie die Plattform- und Programmiersprachenunabhängigkeit der Metasprache XML und basieren nicht auf proprietären sondern etablierten offenen Protokollen. Somit werden die im vorigen Abschnitt aufgezeigten Eigenschaften verstärkt, da sowohl auf jeder Plattform als auch in jeder Programmiersprache Funktionalitäten für die Nutzung von TCP/IP und HTTP sowie XML vorgehalten werden. Dadurch erreichen Web Services einen hohen Grad an Interoperabilität und eine plattformübergreifende Integration in IV-Systeme wird problemlos ermöglicht. Web Services können prinzipiell von jedem digitalen Endgerät aus verwendet werden und besitzen daher einen gewissen Grad an Ubiquität.[15] Diese Eigenschaften fördern auch die Verbreitung und Nutzung von Web Services.[16]

---

[10] Vgl. Bettag (2001), S. 302ff.
[11] Vgl. Hoidn / Jungclaus (2002), S. 33.; Knuth (2002), S. 12.
[12] Vgl. Chappell / Jewell / Dalheimer (2003), S. 80.
[13] Vgl. Austin / Barbir / Ferris / Garg (2002).
[14] Vgl. Hoidn / Jungclaus (2002), S. 32.
[15] Vgl. Wojciechoski / Weinhardt (2002), S. 102.
[16] Vgl. Snell / Tidwell / Kulchenko (2002), S. 6ff.

Im Gegensatz zu dieser eher technischen Sichtweise auf die Innovation können Web Services auch aus einer ökonomischen Sicht als gekapselte und modulare Geschäftsanwendungen (Business Units) angesehen werden, die eine fest definierte Geschäftsfunktionalität abbilden. Somit stellen Web Services eine neue Möglichkeit dar, so genannte digitale Dienstleistungen über Netzwerke und standardisierte Schnittstellen zu vertreiben. Durch dieses Angebot von digitalen Dienstleistungen können die Informationsflüsse innerhalb aber insbesondere auch außerhalb der Unternehmensgrenzen verbessert werden, da der Dienstnutzer unmittelbar die angebotene digitale Dienstleistung in die vorhandene IT-Landschaft oder einen Geschäftsprozess integrieren kann.[17] Die angebotenen digitalen Dienstleistungen bzw. Geschäftsprozessfunktionalitäten können dabei entweder einzeln verwendet oder zu komplexen Geschäftsabläufen zusammengefügt werden.[18] Dann wird von so genannten Super-Services gesprochen.[19] Web Services schaffen somit eine Möglichkeit, verteilte Systeme zu planen, zu realisieren und zu betreiben.

2.2 Architekturmodell

Bei dem Web Service Architekturmodell handelt es sich um die konkrete Umsetzung der bereits seit Anfang der 70er Jahre diskutierten Service Oriented Architecture (SOA), die allerdings auch teilweise heutzutage noch als neues Paradigma in Rahmen der verteilten Anwendungsentwicklung angesehen wird.[20] Dabei zeichnen sich serviceorientierte Architekturen durch verteilte Softwarekomponenten aus, die eine spezielle Dienstleistung zur Verfügung stellen und die lose miteinander gekoppelt werden können. Die Kommunikation erfolgt über Netzwerke, vorzugsweise über das Internet, und basiert auf offenen Standards.[21] Innerhalb dieser Architektur werden die drei Rollen des Dienstanbieters, des Dienstnutzers und des Dienstmaklers unterschieden, die untereinander interagieren und Nachrichten austauschen. Als Synonym für „Rolle“ kann auch der Begriff des Akteurs verwendet werden. Zentrale Bestandteile bei der Interaktion der drei beteiligten Parteien sind auf der einen Seite der Dienst (Service) und auf der anderen Seite die Dienstbeschreibung.[22] Abbildung 1

[17] Vgl. Fessenbecker (2002), S. 47.; Hars / Schlüter-Langdon (2002), S. 16f.
[18] Vgl. Kreger (2001), S. 6.
[19] Vgl. Chappell / Jewell / Dalheimer (2003), S. 5ff.
[20] Vgl. Mohan (2002), S. 1ff.
[21] Vgl. Hoidn / Jungclaus (2002), S. 33.; Schoder / Fischbach / Teichmann (2002), S. 103.
[22] Vgl. Gottschalk / Graham / Kreger / Snell (2002), S. 170ff.; Rieck / Dostal / Schandl / Sieb (2003), S. 294ff.; Stevens (2002a); Stevens (2002b).

veranschaulicht die drei Rollen, die Interaktionen zwischen den Rollen sowie die zentralen Bestandteile dieser Architektur grafisch.

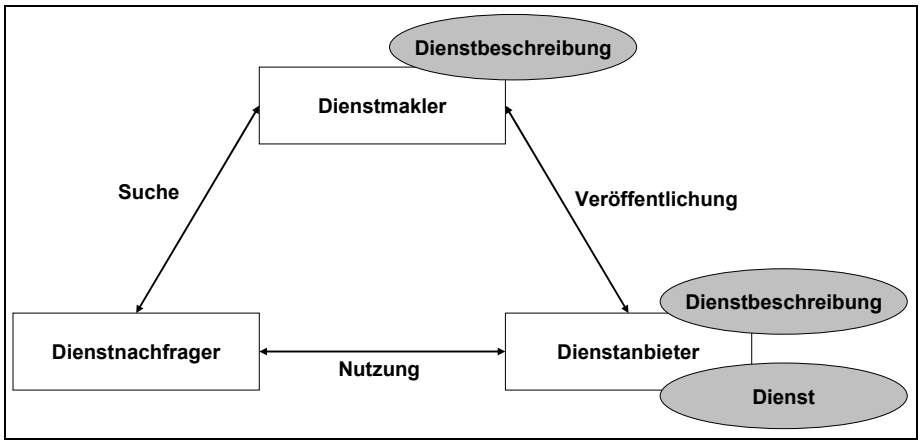

Abbildung 1: Architekturmodell für Web Services

Den drei Rollen des Architekturkonzepts lassen sich die folgenden Aufgaben zuordnen, wobei bei jeder Rolle eine fachliche, eine technische und eine geschäftsorientierte Betrachtungsperspektive dargestellt wird:[23]

- **Dienstanbieter**: Der Dienstanbieter (Service Provider) stellt einen bestimmten Dienst zur Verfügung und beschreibt und annonciert diesen. Der Dienstanbieter übernimmt somit aus der fachlichen Perspektive die Konzeption des Dienstes selbst. Darüber hinaus ist der Dienstanbieter aus technischer Sicht für die Umsetzung des Moduls sowie den Betrieb und die Wartung des Dienstes und der Dienstplattform verantwortlich. Aus einer geschäftsorientierten Betrachtungspunkt ist er somit der Eigentümer des Moduls, der eine bestimmte Funktionalität bzw. Geschäftslogik zur Verfügung stellt, und der Betreiber der Dienstplattform.

- **Dienstnachfrager**: Der Dienstnachfrager (Service Requestor) nutzt den Dienst, indem er ihn in seine Geschäftsprozesse und seine Anwendungen integriert. Aus technischer Sicht handelt es sich um eine Anwendung, die den Dienst aufruft oder mit ihm interagiert. Dabei ist es unerheblich, ob der Dienst direkt durch eine andere Anwendung genutzt wird, oder aber die Funktionalität über eine Benutzerschnittstelle angesprochen wird. Im Rahmen der geschäftsorientierten Perspektive stellt der Nachfrager also den Konsumenten des Dienstes dar.

[23] Vgl. Snell / Tidwell / Kulchenko (2002), S. 5ff.; Cerami (2002), S. 9ff.; Rieck / Dostal / Schandl / Sieb (2003), S. 294f.; Kreger (2001), S. 7f.

- **Dienstmakler**: Der Dienstmakler (Service Registry) ermöglicht aus der fachlichen Sicht die Speicherung der Dienstbeschreibungen in einem zentralen Verzeichnis. Darüber hinaus ermöglicht der Dienstmakler dem Dienstanbieter die Aktualisierung der hinterlegten Informationen sowie dem Dienstnutzer die Suche nach entsprechenden Diensten. Auch kann der Dienstnutzer durch den Dienstmakler eine Referenz auf die Beschreibung der Dienste erhalten. Um die Suche nach diesen Beschreibungen möglichst performant und flexibel gestalten zu können, können die Dienste und die Dienstanbieter optional anhand von Metadaten kategorisiert werden. Durch diese Kategorisierung werden verschiedene Sichten auf die Dienste und deren Beschreibungen ermöglicht bzw. die Suche nach bestimmten Diensten vereinfacht. Diese abgelegten Dienstbeschreibungen können dann durch den Dienstnachfrager bei der Suche inspiziert werden.[24] Aus der technischen Perspektive handelt es sich um eine Anwendung, welche die für den Dienst zugrunde liegenden fachlichen Anforderungen umsetzt. Aus der geschäftsorientierten Perspektive handelt es sich um ein Unternehmen, welches diese Suchdienste und die Ablagefunktionalitäten zur Verfügung stellt. Somit wird dadurch eine Art Vermittleraufgabe zwischen dem Dienstanbieter und dem Dienstnachfrager übernommen.[25]

Die Interaktion zwischen den drei aufgezeigten Rollen kann unter den Schlagwörtern der Veröffentlichung, der Suche und der Nutzung zusammengefasst werden, die nachfolgend näher erläutert werden sollen:[26]

- **Veröffentlichung**: Damit eine Nutzung des Dienstes möglich wird, muss der Anbieter die Dienstbeschreibung dem Dienstnutzer zukommen lassen. Dazu veröffentlicht er diese Beschreibung beim Dienstmakler, über den der Dienstnachfrager dann diese Informationen bezieht. Neben diesem indirekten Weg der Veröffentlichung, der auch als dynamische Veröffentlichung bezeichnet wird, kann der Dienstanbieter aber auch direkt den Dienstnachfrager über die Dienstbeschreibung informieren, falls dieser bekannt ist. Dieser direkte Weg der Veröffentlichung wird auch als statische Veröffentlichung des Dienstes bezeichnet. In diesem Zusammenhang wird auch die

[24] Vgl. Seely (2002), S. 197ff.; Snell / Tidwell / Kulchenko (2002), S. 5 u. 105ff.
[25] Vgl. Langner (2003), S. 76f.
[26] Vgl. Beimborn / Mintert / Weitzel (2002), S. 277f.; Bettag (2001), S. 302ff.; Schoder / Fischbach / Teichmann (2002), S. 103.; Kreger (2001), S. 8.

Komplementärfunktion der Löschung der Veröffentlichung eingegliedert, die anfällt, wenn ein Dienstangebot durch den Dienstanbieter zurückgezogen wird.

- **Suche**: Die Operation der Suche fällt nur im Rahmen der dynamischen Veröffentlichung der Dienste an. Dabei durchsucht der Dienstnutzer die beim Dienstmakler durch die Dienstanbieter in einem Dienstverzeichnis abgelegten Dienstbeschreibungen. Die Suche wird durch die Kategorisierung der Beschreibungen erleichtert. Bei einer erfolgreichen Suche bezieht der Dienstnachfrager dann die Dienstbeschreibung bzw. einen Verweis auf dieselbige, die alle notwendigen technischen Informationen für die die Dienstnutzung beinhaltet.

- **Nutzung**: Durch die vom Dienstmakler (indirekte Information) oder vom Dienstanbieter (direkter Information) erhaltene Dienstbeschreibung besitzt der Dienstnachfrager nun alle notwendigen technischen Informationen, um den Dienst zu aktivieren und zu nutzen. Diese Interaktion wird als Dienstkommunikation bezeichnet.

Wesentliche Bestandteile dieser Architektur sind somit auf der einen Seite der Dienst, der wie bereits angeführt ein Softwaremodul darstellt, dessen Funktionalität und dessen Schnittstellen durch die Dienstbeschreibung spezifiziert werden. Die zentrale Anforderung an einen Softwarekomponente ist dessen Erreichbarkeit durch den Nachfrager über ein Netzwerk. Darüber hinaus ist es möglich, den Dienst aufgrund der standardisierten Schnittstellen lose mit anderen Anwendungen oder Komponenten zu koppeln und dabei auf unterschiedliche Kommunikationsparadigmen zurückzugreifen. Um eine Lokalisierung zu ermöglichen, werden das Softwaremodul selbst und die zugehörige Dienstbeschreibung durch den Anbieter statisch oder dynamisch veröffentlicht.[27] Durch diese Architektur wird somit eine Trennung der Schnittstellenbeschreibung und der konkreten Implementierung des Dienstes erreicht. Auf der anderen Seite bildet die Dienstbeschreibung einen zentralen Bestandteil, auf deren Basis Informationen über die Komponente wie die Schnittstellenspezifikation, genutzte Datentypen und Bindungsinformationen sowie die Netzwerkadresse übermittelt werden. Darüber hinaus kann die Beschreibung Zusatz-informationen, beispielsweise für eine mögliche Kategorisierung, enthalten. Diese

[27] Vgl. Chappell / Jewell / Dalheimer (2003), S. 7ff.

wesentlichen Bestandteile werden in der Literatur auch als Web Service Artefakte bezeichnet.[28]

Aus diesen Ausführungen lässt sich nun ein Lebenszyklus für einen Web Service ableiten, in dem jeder Akteur spezifische Aufgaben übernimmt. Dabei lassen sich die vier Phasen der Erstellung, der Veröffentlichung, der Nutzung und der Verwaltung unterscheiden. Zu Beginn wird der Dienst in der Phase der Erstellung (Build) durch den Anbieter konzipiert und technisch umgesetzt. Auch wird durch den Dienstanbieter eine Beschreibung verfasst. In der daran anschließenden Phase der Veröffentlichung (Deploy) wird der Dienst beim Dienstmakler registriert. Der Dienstnachfrager informiert sich in der Nutzungsphase (Run) beim Makler bzw. direkt beim Anbieter über die angebotenen Dienste, bezieht die Beschreibung und kann auf deren Basis das Softwaremodul einbinden und nutzen. Am Ende der Laufzeit eines Dienstangebots kann der Dienstanbieter den Dienst zurückziehen, in dem er die Veröffentlichung rückgängig macht und optional die Nutzer über diesen Schritt informiert. Parallel zur Nutzungsphase des Dienstes findet durch den Anbieter die Verwaltung des Dienstes (Manage) statt. In dieser Phase wird beispielsweise die Verfügbarkeit sowie optional die Weiterentwicklung des Dienstes sichergestellt. Dieser Lebenszyklus ist in Abbildung 2 dargestellt.[29]

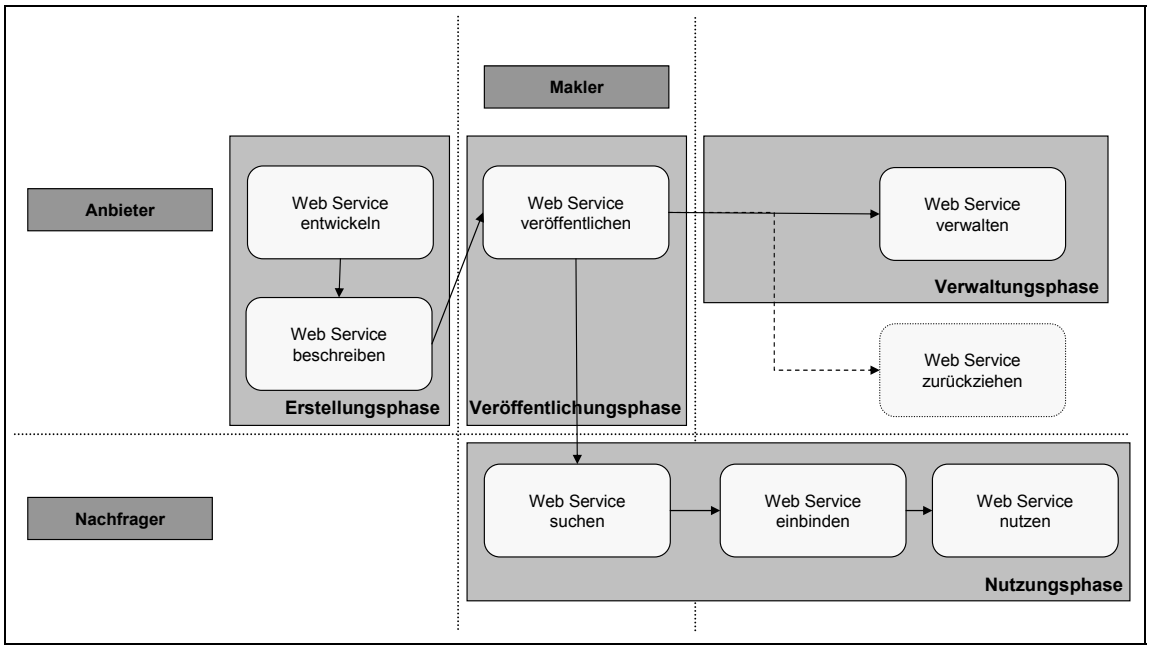

Abbildung 2: Lebenszyklus eines Web Services

Innerhalb des Lebenszyklus erfolgt die gesamte Interaktion zwischen den drei beteiligten Rollen sowie die Nutzung des Dienstes über standardisierte Schnittstellen, die über ein

[28] Vgl. Kreger (2001), S. 8.
[29] Vgl. Burghardt / Gehrke / Schumann (2003a), S. 72f.; Rawolle / Burghardt (2002), S. 40ff.; Jeckle (2002).

Netzwerk angesprochen, dynamisch gefunden und eingebunden werden können.[30] Somit kann dieses Architekturkonzept als eine Weiterentwicklung des traditionellen Client-Server-Konzepts gesehen werden.[31]

2.3 Kommunikationsparadigmen

Im Rahmen des vorgestellten Architekturmodells werden sowohl entfernte Methodenaufrufe (Remote Procedure Calls bzw. Remote Method Invocation) als auch die nachrichtenbasierte Kommunikation (Message Oriented Middleware) als Kommunikationsparadigmen unterstützt und bieten die Basis für die durch Web Services ermöglichte Funktionsintegration. Daher obliegt es den nachfolgenden Abschnitten, diese zu erläutern. Im Fall von Web Services werden die Nutzinformationen sowohl beim Remote Procedure Call als auch bei der Message Oriented Middleware durch XML-Dokumente abgebildet.

2.3.1 Remote Procedure Call/ Remote Method Invocation

Der Remote Procedure Call (RPC) ist in der Informatik bereits seit Anfang der achtziger Jahre bekannt.[32] Er stellt einen Mechanismus dar, der das Aufrufen von Funktionen erlaubt, die sich auf dem gleichen oder einem entfernten Netzwerkknoten befinden können. Dabei sind RPCs gerade bei der Entwicklung von verteilten Anwendungen relativ weit verbreitet, da der entfernte Funktionsaufruf dem Aufruf einer lokalen Funktion bei Verwendung einer höheren Programmiersprache ähnelt.[33] Remote Procedure Calls stellen ein sehr systemnahes Konzept zur Funktionsintegration dar.[34] Um der zunehmenden Nutzung von objektorientierten Programmiersprachen Rechnung zu tragen, wurde der Mechanismus des entfernten Funktionsaufrufs angepasst und wird im objektorientierten Umfeld nun als Remote Method Invocation (RMI) bezeichnet. RMI ermöglicht den entfernten Methodenaufruf anderer Objekte.[35] Die Unterschiede zwischen RPC und RMI sind jedoch minimal, so dass nachfolgend keine Unterscheidung der beiden Konzepte vorgenommen wird.

Bei RPC und auch bei RMI werden die entfernten Aufrufe in Request- und Reply-Nachrichten im Sinne des Client-Server-Kommunikationsmodells gekapselt. Dabei läuft der

---

30 Vgl. Löwer / Picot (2002), S. 21ff.
31 Vgl. Sleeper / Robins (2002).
32 Vgl. Badach / Rieger / Schmauch (2003), S. 312f.
33 Vgl. Birrell / Nelson (1984), S. 39ff.
34 Vgl. Kurbel / Rautenstrauch / Rödding / Scheuch (1995), S. 449f.
35 Vgl. Coulouris / Dollimore / Kindberg (2002), S. 203f.

entfernte Methodenaufruf wie folgt ab. Der aufrufende Prozess sendet eine Anfragenachricht (Request-Nachricht), die den entfernten Funktionsaufruf kapselt, über das Netzwerk an den entfernten Prozess. Dabei enthält die Nachricht neben den Funktionsnamen die notwendigen Parameter, die für den erfolgreichen Aufruf der Funktion benötigt werden. Der aufgerufene Prozess verarbeitet die Anfrage und generiert eine geeignete Antwortnachricht (Reply-Nachricht). Diese wird zum aufrufenden Prozess übermittelt, der auf die Antwortnachricht wartet und somit während der Bearbeitungsdauer der Anfrage durch den aufgerufenen Prozess blockiert ist. Der Informationsaustausch sowie das Nachrichtendesign werden durch eine spezielle RPC-Middleware realisiert, die für den Anwender Application Programming Interfaces (APIs) bereitstellt.[36] Das API stellt eine abstrakte Schnittstelle dar, die den Funktionsaufruf und somit die Kommunikation kapselt. Abbildung 3 verdeutlich diesen Ablauf graphisch.

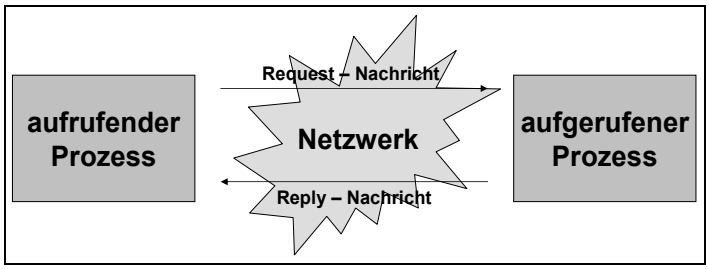

Abbildung 3: Grundprinzip des Remote Procedure Calls (RPC)

Prinzipiell kann neben dem hier vorgestellten synchronen Funktionsaufruf, der in der Praxis in den meisten Fällen Anwendung findet, auch ein asynchroner bzw. ein asynchroner gepufferter Funktionsaufruf erfolgen. Beim asynchronen Funktionsaufruf führt der aufrufende Prozess nach Übermittlung der Anfrage-Nachricht weitere Aktionen auf, ohne auf die Antwort des aufgerufenen Prozesses zu warten. Beide Prozesse arbeiten also parallel. Die Antwortnachricht wird dem aufrufenden Prozess dann über ein Ereignis zurückgemeldet. Beim asynchronen gepufferten Methodenaufruf werden die Nachrichten in einem Puffer abgelegt, der von den beteiligten Prozessen in regelmäßigen Abständen Nachrichten abgefragt wird.[37]

Während der synchrone RPC das Grundprinzip einer entfernten Funktionsaufrufs beinhaltet, stellen die letzteren beiden Formen Erweiterungen des Grundprinzips dar, die aus der Anforderung nach mehr Flexibilität und Effizienz an das diskutierte Kommunikationsparadigma entstanden sind.

---

[36] Vgl. Haase (2001), S. 8ff.
[37] Vgl. Hansen / Neumann (2002), S. 167ff.

2.3.2 Message Oriented Middleware

Im Gegensatz zum Remote Procedure Call (RPC) kommunizieren die Prozesse bei der Message Oriented Middleware (MOM) ebenfalls über den Austausch von Nachrichten miteinander, jedoch beinhalten diese Nachrichten neben Spezifikationen und Übergabeparametern zu der auszuführenden Methode noch weitere Informationen und Daten wie beispielsweise Kontextinformationen und Kontrollinformationen. Spezielle MOM-Middleware stellt analog zum Remote Procedure Call dem Anwender Application Programming Interfaces (API's) zu Verfügung, die sich um den Nachrichtentransfer zwischen den beteiligten Prozessen kümmern.[38]

Das Nachrichtendesign obliegt dem Anwender selbst und wird nicht durch vorgehaltene Funktionalitäten innerhalb der Middleware geregelt. Dadurch ist MOM hinsichtlich des Nachrichtenlayouts nicht beschränkt und sehr flexibel, da diese Aufgabe durch den Anwender umgesetzt wird. Beim RPC wird hingegen das Nachrichtenlayout durch die entsprechende Middleware dem Anwender gegenüber verschattet. Dies führt zur einer Einschränkung bzgl. der Flexibilität des Nachrichtenlayouts. Somit ergibt sich in diesem Freiheitsgrad im Nachrichtenlayout auch der Hauptunterschied zwischen RPC und MOM.

Analog zum RPC wird durch MOM auch neben der direkten und somit synchronen Kommunikation auch eine indirekte und asynchrone Kommunikation über Warteschlangen ermöglicht. Diese Warteschlange empfängt die Nachricht und speichert diese zwischen, bis der adressierte Prozess zur Verarbeitung bereit ist. Dadurch wird die Blockierung des sendenden Prozesse bis zum Erhalt der Antwortnachricht vom adressierten Prozess vermieden. Die Warteschlange wird durch einen so genannten Warteschlangenmanager (queue manager) verwaltet.[39]

2.4 Einordnung von Web Services in verwandte Konzepte

Im nachfolgenden Abschnitt findet eine Einordnung von Web Services in die Konzepte zur Realisierung verteilter Anwendungen auf der einen Seite und Middleware auf der anderen Seite statt. Diese Einordnung zeigt Gemeinsamkeiten sowie Unterschiede zwischen existierenden Konzepten und der Innovation Web Services auf.

---

[38] Vgl. Britton (c2001), S. 34f.
[39] Vgl. Simon E. (1996), S. 60ff.

2.4.1 Web Services und Architekturen für verteilte Systeme

Web Services bieten die Möglichkeit, verteilte Systeme zu realisieren. Allerdings gehen dabei die Meinungen bzgl. der Eignung dieser Innovation in diesem Aufgabengebiet weit auseinander. So sprechen einige Autoren von einem Paradigmenwechsel während andere Web Services als eine Modeerscheinung bezeichnen.[40]

Zu den etablierten Architekturen, die ebenfalls die Realisierung von verteilten Anwendungen gestatten, zählen die Common Object Request Broker Architecture (CORBA), Suns Java RMI sowie das Distributed Common Object Model (DCOM) von Microsoft. Diese drei Konzepte sollen nun der Innovation Web Services gegenübergestellt werden.

Common Object Request Broker Architecture (CORBA)

Grundlage für die Spezifikation der Common Object Request Broker Architecture bildet die der Object Request Broker (ORB). Der ORB stellt eine Infrastruktur für die Kommunikation und Lokalisierung verteilter Objekte bereit. Federführend war an der Entwicklung die Object Management Group (OMG) beteiligt, die neben der Spezifikation des ORB auch die Spezifikation der Object Management Architecture (OMA) verabschiedeten. Dabei beschränkt sich die OMA lediglich auf die Erarbeitung von Spezifikation und Rahmenbedingungen, die durch konkrete Produkte abgebildet werden müssen.[41]

Die beiden verabschiedeten Spezifikationen CORBA und OMA der OMG verhalten sich komplementär zueinander. CORBA ermöglicht die Kommunikation zwischen Objekten und bildet somit die Grundlage für die OMA. Zentrales Element der CORBA Spezifikation ist der ORB, der als Vermittlungszentrale zwischen den beteiligten Objekte dient und die Kommunikationsabwicklung übernimmt.[42] Dabei werden die Objekte durch eine Schnittstelle, die durch die Interface Definition Language (IDL) beschrieben wird, vom ORB separiert. Die Schnittstellenbeschreibung in Form einer IDL lässt sich unter Anwendung eines IDL-Compilers in jede beliebige Programmiersprache überführen.[43] Die Übertragung der Daten zwischen den Objekten erfolgt in binärer Form (CDR).[44] Die Interaktion der Objekte mit dem ORB erfolgt über ein Application Programming Interface (API), das ebenfalls für beliebige Programmiersprachen zur Verfügung steht. Der ORB kapselt gegenüber dem Objekt die Spezifika der genutzten Hardware, des Betriebssystem sowie des Netzwerkprotokolls.[45]

---

[40] Vgl. Chappell / Jewell / Dalheimer (2003), S. 9f.
[41] Vgl. Steckermeier (2001), S. 31.
[42] Vgl. Mahmoud (2000), S. 153f.; Balen / Elenko (2000), S. 5f.
[43] Vgl. Krüger / Deutschmann (2002), S. 238ff.
[44] Vgl. Müller-Stewens / Eymann / Kreutzer (2003), S. 147.
[45] Vgl. Hartmann / Flinn / Beznosov / Kawamoto (2003), S. 176ff.

Die OMA bildet die Grundlage für unternehmensweite komplexe Anwendungen und baut auf der Spezifikation von CORBA auf, die die technischen Grundlagen der OMA abbildet. Die OMA stellt somit ein Rahmenwerk dar und umfasst neben den anwendungsspezifischen Applikationsobjekten CORBA-Services und CORBA-Facilities. CORBA-Services umfassen Dienste, die die grundlegende Funktionalität des ORB's erweitern, jedoch trotzdem generisch genug sind, um von verschiedenen Applikationsobjekten verwendet werden zu können (z.B. Sicherheit, Transaktionsmanagement, Namensservice). Die CORBA-Facilities hingegen dagegen erweitern den ORB um Funktionalitäten spezieller Anwendungsgebiete wie etwa der Telekommunikationsbranche.[46]

Abbildung 4 stellt die OMA graphisch dar.

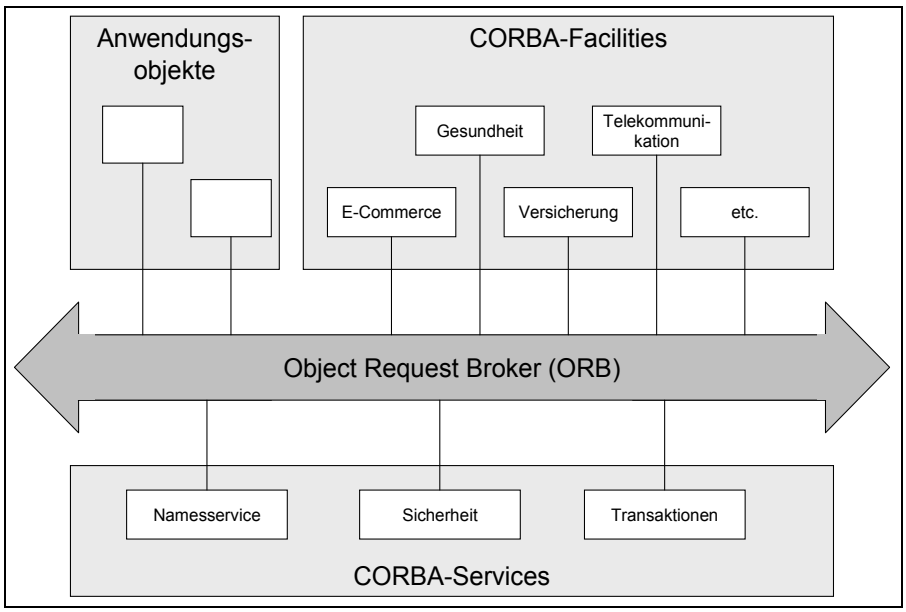

Abbildung 4: Object Management Architecture (OMA)

Java RMI

Java Remote Method Invocation (Java RMI) wurde von Sun Microsystems entwickelt und dient als Infrastruktur für die Entwicklung verteilter Anwendungen auf Basis der Programmiersprache Java. Dabei stellt RMI, wie bereits dargestellt, eine Erweiterung des traditionellen Remote Procedure Calls dar, der die Kommunikation von verteilten Objekten ermöglicht. Dabei sind eine Vielzahl von Objektprotokollen für die Übertragung denkbar, jedoch beschränkt sich Java RMI in der Praxis auf die Nutzung des Java Remote Method Protocol (JRMP) als Objektprotokoll.[47] Diese Kombination wird auch als „JavaRMI over JRMP" bezeichnet. Die Übertragung der relevanten Daten zwischen den Objekten erfolgt in binärer Form. Ebenfalls ist ein direkter Zugriff auf einen ORB möglich, in dem als Objekt-

[46] Vgl. Siegel (1999), S. 120ff.; Krüger / Deutschmann (2002), S. 235ff.
[47] Vgl. Monson-Haefel / Kalle Dalheimer (2001), S. 71ff.

protokoll das Internet Inter-ORB-Protokolls (IIOP) genutzt wird. Eine programmiersprachen-unabhängige Beschreibung der Schnittstelle entfällt, da Java RMI lediglich in Verbindung mit der Programmiersprache Java nutzbar ist. Die Beschreibung erfolgt hier in einer Interface-Definition.[48] Neben einem Namensservice zum Auffinden der Objekte sind bei Java RMI keine weiteren Mehrwertdienste umgesetzt.

Distributed Common Object Model (DCOM)

Das Distributed Common Object Model (DCOM) ist eine von Microsoft entwickelte Erweiterung des Component Object Models (COM), um die Kommunikation verteilter Anwendungen über Systemgrenzen mittels Object Remote Procedure Calls (ORPC) zu unterstützen.[49] Dabei sind der Begriff des ORPC als Synonym für RMI von Microsoft verwendet. Ein Client, der über das Netzwerk auf eine Funktionalität zugreift, kommuniziert über ein Interface mit dem Objekt, welches die angeforderte Leistung erbringt. Dabei wird das Interface durch einen Globally Unique Identifier (GUID) eindeutig identifiziert.[50] Die Übertragung der Informationen erfolgt in binärer Form. Die Schnittstellenbeschreibung erfolgt auf Basis einer speziell für DCOM entwickelten Interface Definition Language von Microsoft (MIDL).[51]

DCOM/COM ist prinzipiell unabhängig von der Programmiersprache und dem Betriebssystem nutzbar. Da jedoch die für den ORPC genutzten Funktionalitäten von DCOM/COM bereits in die Plattformen der Windows Familie eingebunden sind, weist DCOM dort die größte Verbreitung auf. Es sind jedoch auch vereinzelt Produkte für die Plattformen Linux, Unix und Mainframes verfügbar, wie beispielsweise das EntireX Produkt des Herstellers Software AG. Eine echte Plattformunabhängigkeit ist dennoch in der Praxis nicht gegeben.[52]

Gegenüberstellung

CORBA, Java RMI, DCOM und Web Services nutzen das bereits vorgestellte Kommunikationsparadigma des Remote Procedure Call in objektorientierter Form. Jedoch ist bei allen Ansätzen auch die nachrichtenorientierte Kommunikation möglich. Dabei werden auf Client- und Serverseite Funktionalitäten vorgehalten, die die entsprechenden Anfrage- und Antwortnachrichten zusammensetzen und die Kommunikation zwischen dem Client und dem Server abwickeln. Obwohl alle Konzepte diese Funktionalitäten bereitstellen, ist die Namensgebung bei den einzelnen Ansätzen nicht einheitlich.

---

[48] Vgl. Langner (2002), S. 107ff.
[49] Vgl. Wall / Lader (c2002), S. 11.
[50] Vgl. Krüger / Deutschmann (2002), S. 243ff.
[51] Vgl. Krüger / Deutschmann (2002), S. 248.
[52] Vgl. Hartmann / Flinn / Beznosov / Kawamoto (2003), S. 188ff.

Abbildung 5 veranschaulicht diese unterschiedliche Benennung der Funktionalitäten auf Client und Serverseite.

Funktionalität / Technologie	Clientseitig	Serverseitig
CORBA	Stub	Skeleton
Java RMI	Stub	Skeleton
DCOM	Proxy	Stub
Web Services	Service Proxy	Service Implementation Template

Abbildung 5: Bezeichnung der Funktionalitäten auf Client und Serverseite bei RPC

Neben diesen Abweichungen in der Namensgebung der auf Client- und Serverseite vorgehaltenen Funktionalitäten sind weitere Unterschiede in den Konzepten erkennbar, die Einfluss auf die Interoperabilität haben und diese verschlechtern. Dabei lassen sich die drei folgenden wesentlichen Unterschiede ausmachen[53]:

- **Namensgebung der Verbindungsendpunkte:** Die vorgehaltene Funktionalität eines Dienstes lässt sich unter Verwendung des Verbindungsendpunktes ansprechen. Ein Verbindungsendpunkt ist somit in der Lage, Nachrichten zu empfangen, zu verarbeiten und optional Nachrichten zu versenden. Dieser Verbindungsendpunkt wird in CORBA durch eine Interoperable Object Reference (IOR) identifiziert. Bei DCOM wird hingegen eine Object Reference (OBJREF) verwendet, die sich unter anderem aus der bereits aufgeführten Globally Unique Identifier (GUID) zusammensetzt. Beide Namensgebungen verweisen auf den Verbindungsendpunkt, sind jedoch unterschiedlich aufgebaut. Java RMI und Web Services nutzen als Adressinformation den Standard des Uniform Resource Identifier (URI) bzw. den Uniform Resource Locator (URL) und sind somit prinzipiell ähnlich aufgebaut.
- **Unterstützung mehrerer Schnittstellen (Interfaces) pro Objekt:** CORBA und Java RMI unterstützen einfache implizierte Schnittstellen (Interfaces) pro Objekt, hingegen erlauben DCOM und Web Services mehrere Schnittstellen pro Objekt vor.
- **Format der übertragenen Nachricht:** Alle Konzepte nutzen unterschiedliche Verpackungen der zu übertragenden Anfrage- und Antwortnachrichten. CORBA nutzt die Common Data Representation (CDR), DCOM setzt die Network Data Representation (NDR) ein und Java RMI nutzt Java MarshalledObjects.[54] Diese Übertragsformate transportieren die Informationen in Binärform. Hingegen verwenden

[53] Vgl. Krüger / Deutschmann (2002), S. 233ff.; Langner (2002), S. 107ff.
[54] Vgl. Müller-Stewens / Eymann / Kreutzer (2003), S. 147.

Web Services XML als Nachrichtenformat, die Codierung erfolgt dementsprechend in einer alphanumerischen und somit menschenlesbaren Form.

Im Vergleich zu CORBA, Java RMI und DCOM setzen Web Services sowohl bei der Bezeichnung der Endpunkte als auch beim Nachrichtenformat sowie bei der Schnittstellenbeschreibung auf offene Standards. So ist es möglich, mit etablierten Internetprotokollen (z.B. HTTP) und der Metasprache XML zwei wesentliche Nachteile der proprietären Standards von CORBA, Java RMI und DCOM zu eliminieren und die Anwendung erheblich zu vereinfachen. Abbildung 6 zeigt die eben dargestellten Unterschiede der einzelnen Konzepte tabellarisch auf.

Attribut / Technologie	Endpunktbezeichner	Interfaces pro Objekt	Nachrichtenformat
CORBA	IOR	Single	CDR
Java RMI	URL	Single	Java Marshalled Object
DCOM	OBJREF	Multiple	NDR
Web Services	HTTP-based URL	Multiple	XML

Abbildung 6: Unterschiede in den Konzepten zur Realisierung von verteilte Anwendungen

Bewertung

Aus der Begriffsdefinition von Web Services ergeben sich eine Reihe von Eigenschaften, durch die sich die Innovation Web Service auszeichnet. Diese Eigenschaften können mit den Begriffen Plattform-, Programmiersprachen- und Protokollunabhängigkeit hinsichtlich des Übertragungsprotokolls, Lesbarkeit der Nachrichten, Objektorientierung und Möglichkeit der Beschreibung der Schnittstellen bezeichnet werden. Daher soll nachfolgender Vergleich die Unterschiede der einzelnen Konzepte in Bezug auf diese essentiellen Merkmale von Web Services aufzeigen.

Bezogen auf die Plattformunabhänigkeit, so sind die Konzepte CORBA, Java RMI und Web Services auf allen Betriebssystemen nutzbar. Lediglich DCOM nutzt Funktionalitäten, die standardmäßig nur in Produkte der Windows Plattform eingebunden sind, Anbieter für andere Plattformen sind nur vereinzelt zu finden.[55] Daher ergibt sich für DCOM eine eingeschränkte Plattformunabhängigkeit.

Hinsichtlich der Programmiersprachenunabhängigkeit sind die Konzepte CORBA, DCOM und Web Services mit beliebigen Programmiersprachen realisierbar, lediglich Java RMI beschränkt sich auf die Programmiersprache Java.

[55] Vgl. Hartmann / Flinn / Beznosov / Kawamoto (2003), S. 188ff.

Im Bereich der Protokollunabhängigkeit hinsichtlich des Übertragungsprotokolls weisen Web Services die größte Flexibilität auf, da die Nachrichten an beliebige Übertragungsprotokolle gebunden werden können. Bei CORBA und Java RMI haben sich, wie bereits aufgezeigt, mit IIOP und JRMP als Kommunikationsstandards durchgesetzt, allerdings ist der Einsatz weitere Übertragsprotokolle möglich.[56] Bei DCOM werden die möglichen Übertragsprotokolle allerdings durch das Betriebssystem eingeschränkt, so dass durch den Anwender kein Einfluss auf diese ausgeübt werden kann. Somit ist die Flexibilität dort am geringsten.

Wie bereits angesprochen ist die Lesbarkeit der Nachrichten bei CORBA, Java RMI als auch DCOM nicht vorhanden, da bei diesen Konzepten eine binäre Übertragung der Nachrichten stattfindet. Hingegen nutzen Web Services die Metasprache XML zur Strukturierung der Nachrichten, wodurch eine gute Lesbarkeit der Nachrichten erreicht wird.[57]

Bei allen vorgestellten Konzepten wird die Kommunikation zwischen Objekten durch den Austausch von Nachrichten ermöglicht. Allerdings erhält man bei CORBA, DCOM und Java RMI eine Referenz auf das Objekt, so dass bei einem Methodenaufruf nicht immer alle Initialisierungsinformationen der angesprochenen Objekte mit übertragen werden müssen. Dies ist bei Web Services nicht der Fall, da Web Services zustandlos sind. Somit müssen immer alle relevanten Informationen als Parameter, also auch die Initialisierungs-informationen, übergeben werden.[58] Daher ist die Objektorientierung bei CORBA, DCOM und Java RMI gut ausgeprägt, bei Web Services ist die Objektorientierung nur eingeschränkt vorhanden. Allerdings erlauben alle Konzepte die Übertragung von kompletten Objekten zwischen den Kommunikationspartnern.

Die Schnittstellenbeschreibung erfolgt bei Web Services in Form von XML-Dokumenten, wobei die Dokumente zusätzlich mit Metainformationen angereichert werden können. Bei CORBA und DCOM erfolgt die Schnittstellenbeschreibung durch eine Interface Definition Language (IDL). Alle drei Konzepte ermöglichen somit eine Umsetzung der Beschreibungen in beliebigen Programmiersprachen, da die Schnittstellenbeschreibung von der Implementierung getrennt wurde. Die Möglichkeit der Schnittstellenbeschreibung fällt jedoch bei Web Services aufgrund der Möglichkeit, optional Metadaten anzureichern, besser aus. Bei Java RMI erfolgt die Beschreibung auf Basis von Interfaces, die ebenfalls in der Programmiersprache Java verfasst werden. Dies führt zu einer sehr eingeschränkten Beschreibungsmöglichkeit der Schnittstelle. Somit ist die Beschreibungsmöglichkeit der Schnittstelle bei Java RMI im Vergleich zu den anderen Konzepten mit gering zu bewerten.

---

[56] Vgl. Monson-Haefel / Kalle Dalheimer (2001), S. 71ff.
[57] Vgl. Müller-Stewens / Eymann / Kreutzer (2003), S. 147.
[58] Vgl. Hoidn / Jungclaus (2002), S. 32f.

Abbildung 7 fasst die Ergebnisse der Gegenüberstellung in einer Übersicht zusammen.

Technologie / Eigenschaft	CORBA	Java RMI	DCOM	Web Services
Plattform-unabhängigkeit	+	+	-	+
Programmiersprachen-unabhängigkeit	+	-	+	+
Übertragungsprotokoll-unabhängigkeit	O	O	-	+
Lesbarkeit der Nachrichten	-	-	-	+
Objektorientierung	+	+	+	O
Schnittstellen-beschreibung	O	-	O	+

Abbildung 7: Vergleich von Technologien für verteilte Anwendungen

2.4.2 Web Services und Middleware

Im Folgenden wird zuerst eine Einführung des Begriffs Middleware vorgenommen und mögliche Ausprägungsarten von Middleware aufgezeigt. Daran anschließend werden Web Services einer Art von Middleware zugeordnet.

Middleware hat die Aufgabe, eine einheitliche Schnittstelle zwischen verschiedenen Komponenten eines verteilten Systems zu schaffen, über die beteiligte Systemteile Nachrichten austauschen können. Dabei lässt sich Middleware als Verbindungsschicht zwischen den Systemen auffassen. Middleware bedient sich hauptsächlich dem Prinzip der Kapselung, um technische Details einer Komponente hinter eine generischen Schnittstelle zu verbergen.[59]

Bevorzugtes Einsatzgebiet von Middleware ist die Client-Server-Architektur. Neben der Kommunikation ermöglicht Middleware den Clients Zugriff auf so genannte Dienstleistungen („Services“), die der Server bereitstellt. Abstrakt kann Middleware somit zwischen dem Anwendungsprogramm auf dem Client und den Services auf Serverseite angesiedelt werden.[60] Abbildung 8 veranschaulicht diesen Sachverhalt überblicksartig.

[59] Vgl. Müller-Stewens / Eymann / Kreutzer (2003), S. 146.
[60] Vgl. Orfali / Harkey / Edwards (1997), S. 42.

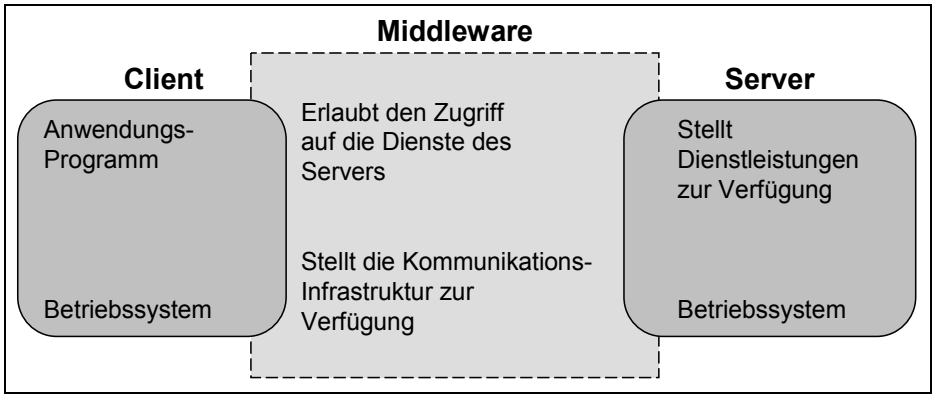

Abbildung 8: Schematische Einordnung von Middleware in eine Client-Server-Umgebung

Middleware lässt sich anhand ihrer Aufgabe in Datenbank-Middleware, zugreifende Middleware und zugangsschaffende Middleware unterteilen.

Datenbank-Middleware ermöglicht den Zugriff auf verschiedene Datenbanken unterschiedlicher Hersteller. Dabei liegt die Herausforderung in der Verbindung verschiedener Zugriffsprotokolle und der Kombination der benötigten Daten aus teilweise verteilten Datenbeständen. Daher lassen sich in diesem Bereich Datenbank-Schnittstellen von Datenbank-Gateways unterscheiden, wobei DB-Schnittstellen eine offene, standardisierte Schnittstelle für den Datenbankzugriff zu Verfügung stellen, die allerdings auf herstellerspezifische Treiber zurückgreift. DB-Gateways stellen hingegen eine einheitliche herstellerunabhängige Schnittstelle zur Verfügung. Da sie einen herstellerunabhängigen Zugang gewährleisten sollen, ist der Funktionsumfang bei DB-Gateways im Vergleich zu DB-Schnittstellen geringer und umfasst meistens nur eine Teilmenge der möglichen Funktionalitäten der angebundenen Datenbanksysteme.[61]

Unter zugangsschaffender Middleware werden Mechanismen verstanden, die den Zugang zu Funktionalitäten von Altsystemen ermöglichen und somit dessen Wiederverwendung in neuen Systemkonzeptionen ermöglichen. In diesem Bereich lassen sich Screenscraper und Wrapper unterscheiden. Screenscraper sind Softwarewareprodukte, die eine von einem Großrechner erzeugte alphanumerische Maske einlesen und die darin enthaltenen Informationen extrahieren, um diese in eine graphische Anzeige (GUI) zu überführen oder der Weiterverarbeitung zur Verfügung stellen. Wrapper hingegen sind objektorientierte Softwareprodukte, die Anwendungen auf Basis von prozeduralen Code kapseln und in eine objektorientierte Struktur überführen. Dadurch können diese in einer objektorientierten Systemarchitektur ohne Modifikation der prozeduralen Anwendung integriert und

[61] Vgl. Orfali / Harkey / Edwards (1994), S. 191ff.

weiterverwendet werden. Es wird also eine Kommunikation zwischen Systemen ermöglicht, die auf unterschiedlichen Programmierparadigmen beruhen.[62]

Zugreifende Middleware hingegen gestattet die Nutzung von Dienstleistungen, die ein Server bereitstellt und die durch einen Client in Anspruch genommen wird. Dabei soll diese Klasse von Middleware unabhängig von der genutzten Programmiersprache und der eingesetzten Plattform eine flexible Kommunikation zwischen den zwei Anwendungsinstanzen (Server und Client) ermöglichen.[63] In diesem Zusammenhang werden die grundlegenden Konzepte des Remote Procedure Calls, die nachrichtenorientierte Middleware, der Object Request Broker sowie die Service Oriented Architecture unterschieden. Dabei stellen die ersten beiden Konzepte Kommunikationsparadigmen innerhalb zugreifender Middleware dar, die bereits im vorigen Abschnitt ausführlich dargestellt wurden. Bei den beiden letzten Konzepten handelt es sich um Vermittlungskonzepte, die das Auffinden und die Nutzung der Dienstleistungen ermöglichen.

Abbildung 9 zeigt die Systematisierung von Middleware graphisch auf.

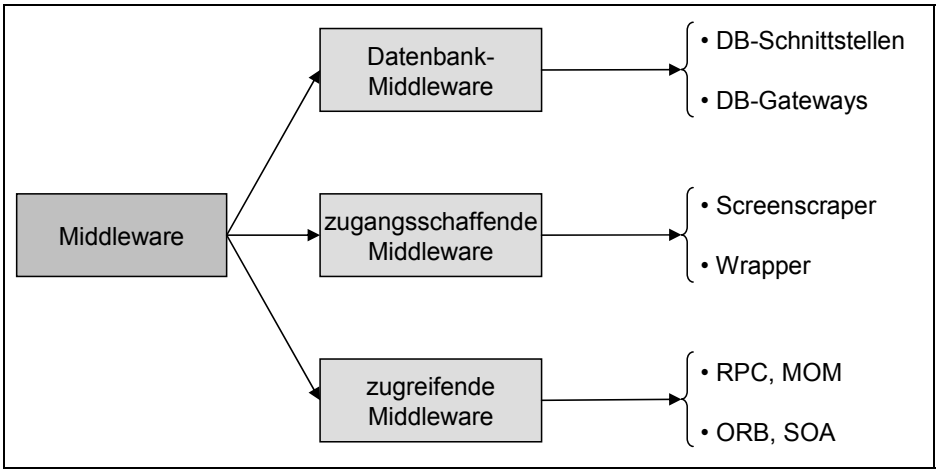

Abbildung 9: Systematisierung von Middleware

Die Ausführungen verdeutlichen, dass Web Services in den Bereich der zugreifenden Middleware eingeordnet werden können. Web Services ermöglichen die programmiersprachen- und plattformunabhängige Kommunikation zwischen zwei Anwendungsinstanzen und nutzen dabei etablierte Internettechnologien. Dabei werden, wie bereits aufgezeigt, durch Web Services beide Kommunikationsparadigmen umgesetzt. Web Services stellen eine konkrete Umsetzung der Service Oriented Architecture dar.

---

62 Vgl. Hoch (1996), S. 152ff.; Itter (1999), S. 48ff.
63 Vgl. Badach / Rieger / Schmauch (2003), S. 322f.

2.5 Schlüsselfaktoren für den Einsatz von Web Services

Aus den bisherigen Ausführungen zum Architekturmodell und den grundlegenden Eigenschaften von Web Services sowie der Bewertung von Konzepten zu verteilten Systemen im vorigen Abschnitt lassen sich nun Schlüsselfaktoren ableiten. Diese Schlüsselfaktoren dienen als Bewertungskriterien und ermöglichen somit eine Urteil darüber, ob Web Services als Schnittstellentechnologie für das Angebot einer digitalen Dienstleistung oder einer Geschäftsprozessfunktionalität eignen oder nicht. Außerdem werden Problemfelder aufgezeigt, die gelöst werden müssen, damit eine weitere Verbreitung und Etablierung von Web Services vorangetrieben wird. Nachfolgend werden diese Schlüsselfaktoren für den Einsatz von Web Services erläutert.

Als ein Schlüsselfaktor kann die Nutzung etablierter Internettechnologien angesehen werden. Im Rahmen von Web Services werden neben dem TCP/IP als Transportprotokoll eine Vielzahl von darauf aufsetzenden Anwendungsprotokollen wie beispielsweise HTTP und SMTP unterstützt, die bereits eine weite Verbreitung und Nutzung durch das Internet erfahren haben.[64] Daher sind Web Services besonders gut geeignet, wenn man auf etablierte Techniken zugreifen möchte. Vergleicht man dies mit anderen Schnittstellentechnologien wie CORBA und Java RMI, so nutzen diese zwar auch TCP/IP als Transportprotokoll, allerdings werden proprietäre Standards wie IIOP bei CORBA und JRMP bei Java RMI als darauf aufsetzende Anwendungsprotokolle genutzt.[65]

Aus dieser Nutzung von etablierten Internettechnologien lassen sich die beiden Schlüsselfaktoren der Plattform- und der Programmiersprachenunabhängigkeit ableiten. Da bei der Anwendung von Web Services etablierte Internettechnologien zu Einsatz kommen, sind in jeder Plattform und in jeder Programmiersprache Funktionalitäten vorhanden, die deren Einsatz ermöglichen. Somit sind Web Services besonders gut geeignet, wenn eine hohe Plattform- und der Programmiersprachenunabhängigkeit angestrebt wird. Bei den alternativen Schnittstellentechnologien wie CORBA und Java RMI ist entweder die Installation von Zusatzsoftware nötig, die die Funktionalitäten zur Nutzung bereitstellt, oder aber es liegt eine Beschränkung hinsichtlich der Wahlfreiheit der Programmiersprache vor. Bei DCOM liegt faktisch eine Beschränkung auf Betriebssysteme der Windows Produktfamilie vor, da auf Funktionalitäten zugegriffen wird, die nur auf dieser Plattform standardmäßig verfügbar sind.

Weitere Schlüsselfaktoren lassen sich aus der Nutzung der Metasprache XML ableiten, die die Grundlage für alle Spezifikationen im Umfeld von Web Services bildet. Durch den Einsatz von XML werden die übertragenen Daten durch die Anreicherung von

---

[64] Vgl. Snell / Tidwell / Kulchenko (2002), S. 6ff.
[65] Vgl. Monson-Haefel / Kalle Dalheimer (2001), S. 71ff.

Auszeichnungselementen auf bis zu 800 % ihres ursprünglichen Volumens aufgebläht.[66] Durch diese Vergrößerung des Übertragungsvolumens kann der Datendurchsatz, der durch Web Services erzielt wird, als Schlüsselfaktor angesehen werden. Daher eignen sich Web Services derzeit nur, wenn ein geringer Datendurchsatz beim Einsatz als unproblematisch und somit als unkritisch angesehen werden kann. Bei den Konkurrenzkonzepten CORBA, Java RMI und DCOM erfolgt die Übertragung in einem binäreren Format, wodurch das Übertragungsvolumen deutlich geringer ausfällt. Dieser Schlüsselfaktor verliert jedoch mit der wachsenden Übertragungsbandbreite innerhalb von Netzwerken zunehmend an Bedeutung. Auch schränkt die Nutzung von XML weder die Plattform- noch die Programmiersprachenunabhängigkeit von Web Services ein, da Funktionalitäten für die Verarbeitung von XML-Dokumenten auf allen Plattformen und unter allen Programmiersprachen verfügbar sind.

Aus den bisherigen Ausführungen lässt sich auch die Interoperabilität der Schnittstelle als weiterer Schlüsselfaktor ausmachen. Sowohl durch die Nutzung etablierter Internetprotokolle als auch die Verwendung von XML wird bei Web Services eine hohe Interoperabilität gewährleistet, d.h. da keine Kompatibilitätsprobleme auftreten ist eine plattformübergreifende Integration von Diensten in Fremdsysteme problemlos möglich. Web Services sind prinzipiell auf jedem digitalen Endgerät nutzbar. Dies ist beispielsweise bei CORBA nicht gegeben, da die dem ORB zugrunde liegende Spezifikation bei den verschiedenen Herstellern der Softwareprodukte unterschiedlich ausgelegt wird. Dies führt dazu, dass die ORB Implementierungen unterschiedlicher Hersteller Kompatibilitätsprobleme aufweisen und somit unter Umständen nicht miteinander kommunizieren können. Dies hat bei einer systemübergreifenden Integration einen langwierigen und schwierigen Abstimmungsprozess zur Folge.[67]

Als weitere Eigenschaft wurde die Selbstbeschreibung von Web Services in der Begriffsdefinition herausgestellt. Daher sind Web Services besonders gut geeignet, wenn eine umfangreiche Beschreibung der Schnittstelle angestrebt wird. Bei Web Services können die Schnittstellenbeschreibungen neben der Spezifikation der Funktionalität an sich um zusätzliche Metainformationen angereichert werden, die Zusatzinformationen über die Basisfunktionalität aufnehmen können. Bei CORBA, Java RMI und DCOM erfolgt hingegen die Schnittstellenbeschreibung auf Basis einer IDL oder eines Interfaces, die lediglich die Funktionalität abbilden, zusätzliche Informationen können jedoch nicht abgelegt werden.

Neben diesen Schlüsselfaktoren, die sich aus den Charakteristika von Web Services ableiten lassen, lassen sich weitere Kriterien identifizieren, die Zusatzanforderungen an Web

---

[66] Vgl. Weitzel / Harder / Buxmann (2001), S. 69ff.
[67] Vgl. Wojciechoski / Weinhardt (2002), S. 102.

Services darstellen, die allerdings nicht durch die Begriffsdefinition abgedeckt werden. Diese lassen sich auf der zunehmenden Verbreitung des Internets und der damit zunehmenden Vernetzung sowie aus Wirtschaftlichkeitsargumenten herleiten. Da Web Services Softwaremodule darstellen, deren Funktionalitäten über Netzwerke und somit auch unter Verwendung des Internet in Anspruch genommen werden können, müssen Sicherheitsanforderungen wie Verfügbarkeit, Vertraulichkeit, Zugriffskontrolle, Zurechenbarkeit und Datenintegrität umgesetzt werden. Die zunehmende Vernetzung ermöglicht darüber hinaus firmenübergreifende Prozesse, die in Zukunft durch die Verkettung von Web Services realisiert werden können. Daher muss auch die Möglichkeit geschaffen werden, komplexe Geschäftsprozesse und somit Workflows auf Basis von Web Services abzubilden. Eng mit den beiden eben aufgeführten Kriterien ist auch die Transaktionssicherheit innerhalb dieser komplexen Workflows zu realisieren. Da Web Services als Schnittstelle für das Angebot von digitalen Dienstleistungen genutzt werden können, ist ebenfalls die Möglichkeit der Abrechnung der Dienstleistungen und somit der Nutzung von Web Services sicherzustellen. Die Lösung dieser Eckpunkte kann als essentiell für die weitere Durchsetzung und Verbreitung von Web Services angesehen werden. Derzeit sind lediglich Ansätze von Spezifikationen zur Lösung dieser Zusatzanforderungen in der Diskussion. Web Services eignen sich als Schnittstellentechnologie somit nur gut, wenn diese Zusatzanforderungen als unwichtig eingestuft werden.
Abbildung 10 veranschaulicht die erläuterten Schlüsselfaktoren und die entsprechenden Ausprägungen graphisch.

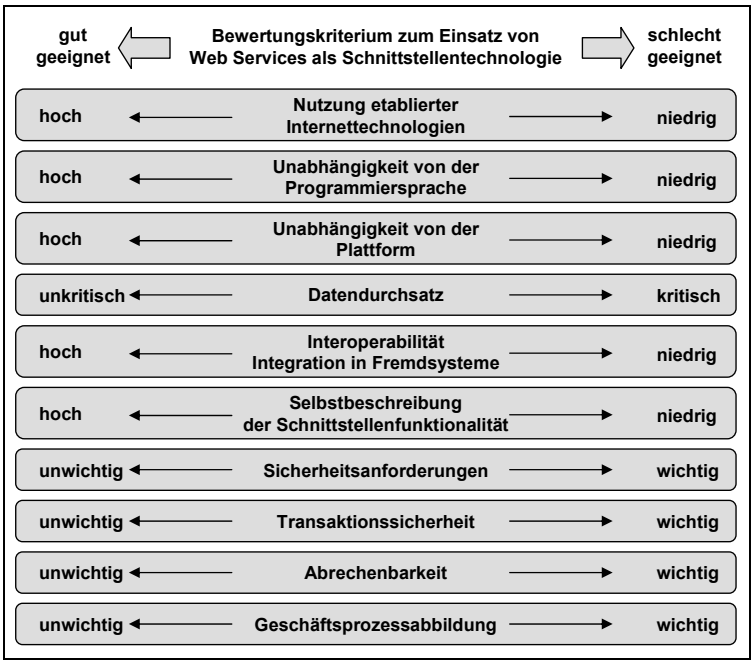

Abbildung 10: Schlüsselfaktoren für den Einsatz von Web Services

3 Basistechnologien

In diesem Abschnitt wird nachfolgend kurz der dem Architekturmodell zugrunde liegende Protokollstapel vorgestellt, mit Hilfe dessen die im Web Services Umfeld existierenden Spezifikationen systematisiert und geordnet werden können. Daran anschließend wird ein Überblick zu verschiedenen Spezifikationen gegeben, die die jeweiligen fachlichen Anforderungen der einzelnen Ebenen abdecken und der jeweils verbreitete Standard, der auch als Kernstandard bezeichnet wird, genauer vorgestellt. Ein Schwerpunkt liegt dabei auf der Metasprache XML, da alle etablierten Standards im Web Services Umfeld auf diese Metasprache zurückgreifen und somit eine Einführung unerlässlich ist. Abschließend werden auch die Möglichkeiten der Werkzeugunterstützung für die Kernstandards diskutiert.

3.1 Web Service Protokollstapel

Um die aufgezeigten Funktionalitäten der Veröffentlichung, der Suche und der Nutzung innerhalb des Architekturmodells und somit innerhalb der Service Oriented Architecture (SOA)[68] zu realisieren, haben sich verschiedene Standards im Web Services Umfeld etabliert, die sich anhand eines Protokollstapels oder eines Zwiebelschalenmodells systematisieren und anordnen lassen. Dabei werden die Ebenen bzw. die Schalen selbst anhand ihrer Funktionalität mit Dienstkommunikation, Dienstbeschreibung und Dienstverzeichnis bezeichnet. Bei der Entwicklung der Standards wurde ein besonderes Augenmerk auf die Interoperabilität gelegt.[69] Diese Schichteneinteilung liegt den Ausführungen zugrunde.

Jede Schicht innerhalb des Protokollstapels realisiert einen Teil der benötigten Funktionalität, die im Architekturmodell vorgesehen sind.[70] Jede Ebene nutzt die bereits realisierten Funktionalitäten der darunter liegenden Schicht und setzt darauf aufbauend neue fachliche Anforderungen des Architekturmodells technisch um. Durch diese Beziehungen zwischen den einzelnen Ebenen wird sukzessive die Funktionalität sowie die Interoperabilität von Web Services erhöht und in Konsequenz daraus die Nutzung eines einzelnen Dienstes zwischen dem Dienstanbieter und dem Dienstnutzer ermöglicht. Auch wird durch die Gewährleistung der Interoperabilität die Austauschbarkeit eines Dienstes ermöglicht, d.h. ein Dienstanbieter

---

[68] Vgl. hierzu die Ausführungen in Abschnitt 2.2
[69] Vgl. Kreger (2001), S. 10ff.
[70] Vgl. Jeckle (2002).

kann die Dienste bei gleicher Schnittstellensignatur problemlos gegeneinander substituieren.[71]

Abbildung 11 veranschaulicht auf der linken Seite die Einordnung der Kernstandards in die Aufgabenbereiche der Dienstkommunikation, der Dienstbeschreibung und der Dienstauffindung anhand des Protokollstapels. Zur Vervollständigung wurde die Transportebene hinzugefügt, die jedoch durch etablierte Internetprotokolle abgedeckt wird und daher im Rahmen dieser Arbeit keine weitere Beachtung findet. Des Weiteren wurde die Schicht der Inhaltsbeschreibung hinzugefügt, da alle Kernstandards auf diese Ebene zurückgreifen und diese daher im Protokollstapel Berücksichtigung finden muss.[72] Auf der rechten Seite ist die Einordnung der Kernstandards im Rahmen des Zwiebelschalenmodells dargestellt. Dort fand die Transportebene zur Wahrung der Übersichtlichkeit keine Berücksichtigung.

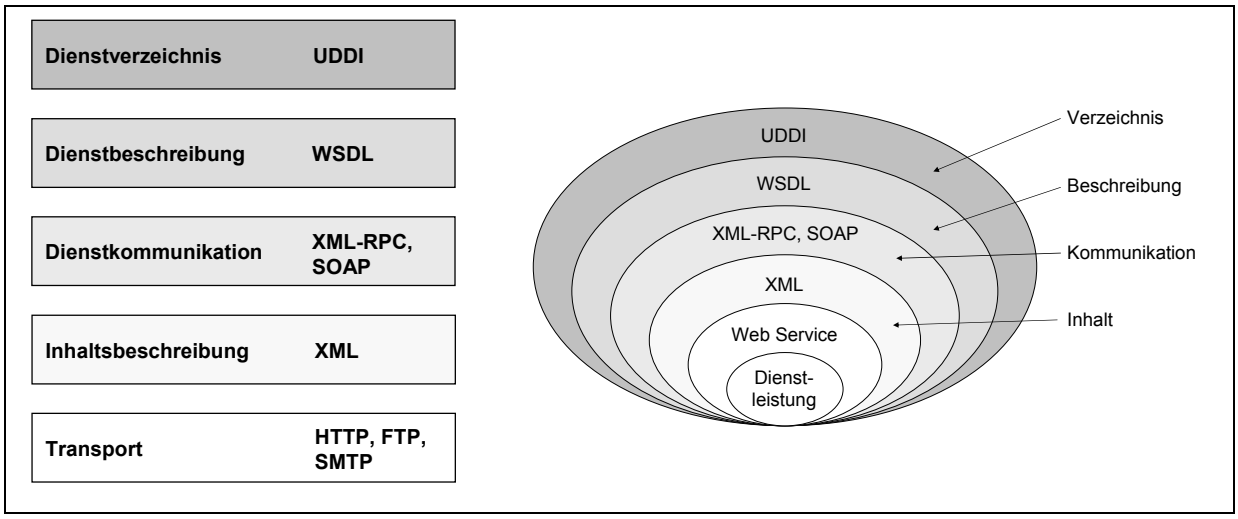

Abbildung 11: Protokollstapel und Zwiebelschalenmodell der Web Service Architektur

3.2 Inhaltsbeschreibung – Extensible Markup Language (XML)

„XML ist eine textbasierte Meta-Auszeichnungssprache, die die Beschreibung, den Austausch, die Darstellung und die Manipulation von strukturierten Daten erlaubt.“[73] Auszeichnungssprachen ermöglichen prinzipiell die Anreicherung von textorientierten Informationen mit Strukturinformationen und Metadaten über so genannte Marken (engl. Tags). Dabei lassen sich unter anderem folgende Auszeichnungsarten unterscheiden:

---

[71] Vgl. Cerami (2002), S. 11ff.; Galbraith / Hankison / Hiotis / Prasad / Trivedi / Whitney D. (2002), S. 9ff.
[72] Vgl. Gottschalk / Graham / Kreger / Snell (2002), S. 170ff.
[73] Weitzel / Harder / Buxmann (2001), S. 18.

- *Semantische Auszeichnungen*: Anreicherung der Inhalte mit Informationen über die Bedeutung der Inhalte
- *Strukturelle Auszeichnungen*: Anreicherung der Inhalte mit Informationen über die Position der Inhalte innerhalb eines Dokuments (Überschriften, Absätze, Referenzen, etc.)
- *Stilistische Auszeichnungen*: Anreicherung der Inhalte mit Informationen über die Formatierung bestimmter Passagen innerhalb eines Dokuments (Schriftgröße, Schriftart, etc.)

Somit stellt XML als Meta-Sprache keine Auszeichungssprache selbst dar, sondern ermöglicht vielmehr die Definition von neuen Auszeichnungssprachen auf Basis von XML.[74] XML kann deshalb in die Klasse der generischen Auszeichnungssprachen eingeordnet werden.[75]

XML wurde durch die XML Working Group, die durch das World Wide Web Consortium (W3C) eingerichtet wurde, entwickelt. Die Verabschiedung als Standard in der Version 1.0 erfolgte im Februar 1998. Als Grundlage des Entwicklungsprozesses diente die Standard Generalized Markup Language (SGML), die bereits seit 1983 in vielen Anwendungsgebieten wie im Verlagswesen und in Software-Systemen eingesetzt wird. Die Weiterentwicklung wurde deswegen fokussiert, da die Komplexität von SGML durch die Nutzer als zu hoch eingestuft wurde. Des Weiteren bildeten die so genannten starren Auszeichnungssprachen wie beispielsweise HTML einen Weiterentwicklungsbedarf, da deren Flexibilität durch die Anwender als unzureichend eingeschätzt wird.[76]

Bei den Entwicklungszielen von XML wurde ein Hauptaugenmerk auf die einfache Handhabung und Beherrschbarkeit von XML gelegt. Darüber hinaus wurde eine Kompatibilität mit SGML und eine hohe Flexibilität bzgl. der genutzten technischen Plattformen und Anwendungsgebiete gefordert. [77]

Als Ergebnis dieses Entwicklungsprozesses unter Einhaltung der Entwicklungsziele entstand die Meta-Sprache XML, die ca. 20 % der Komplexität und ca. 80 % der Funktionalität von SGML umfasst. Jedoch kann XML nicht als ein Nachfolger oder eine Erweiterung der Hypertext Markup Language (HTML) aufgefasst werden, sondern stellt ein Profil von SGML dar. Damit sind alle XML-Dokumente per Konstruktion konforme SGML-Dokumente, so dass

---

[74] Vgl. Phillips (2000), S. 13.
[75] Vgl. Rawolle (2002), S. 52.
[76] Vgl. Genussa (1999), S. 33f.; Priesnitz / Teille (2003), S. 16f.
[77] Für einen Überblick über die Entwicklungsziele der Meta-Auszeichnungssprache XML vergleiche u. a. Bray / Paoli / Sperberg-McQueen / Maler (2000).

XML häufig auch als Teilmenge von SGML bezeichnet wird. Im Gegensatz dazu repräsentiert HTML eine Anwendung von SGML mit einem festgelegten Satz an Marken.[78]

Als Zeichensatz verwendet XML standardmäßig Unicode, sofern keine anderen Angaben innerhalb der Dokumente gemacht werden. Unicode stellt dabei ein plattform-, betriebssystem- und anwendungsübergreifendes Kodierungssystem zur Verfügung, in welchem die Schriftzeichen aller weltweit gesprochenen Sprachen repräsentiert werden. Daher kann XML plattform- und betriebssystemunabhängig eingesetzt werden.[79]

Aufbau eines XML-Dokuments

Der XML-Standard spezifiziert eine Menge von Bestandteilen, die entweder obligatorischen oder optionalen Charakter haben. Zu den obligatorischen Bestandteilen sind die Elemente mit optionalen Attributen sowie der Prolog zu rechnen, zu den optionalen Bestandteilen eines XML-Dokuments zählen Kommentare und Verarbeitungsanweisungen (eng. Processing Instruction).[80]

Ein XML-Dokument beginnt mit dem obligatorischen Prolog. Dieser Prolog besteht dann mindestens aus der Deklaration der genutzten XML-Version. Darüber hinaus können dort auch Angaben über den verwendeten Zeichensatz innerhalb des XML-Dokuments gemacht werden.[81]

Darüber hinaus besitzt ein XML-Dokument mindestens ein Wurzelelement, welches den eigentlich Inhalt des Dokuments umschließt. Ein Element wird durch einen Bezeichner gekennzeichnet und besteht in der Regel aus einer Start- und einer Endmarke, die den eigentlichen Inhalt (weitere Elemente oder Zeichenfolgen) einbetten. Aus dieser Schachtelung ergibt sich insgesamt eine hierarchische Struktur der Dokumente selbst, die auch als Baumstruktur abgebildet werden kann. Optional lassen sich den Elementen Attribute in Form von Attribut-Wert-Paaren zuordnen, die eine genauere Beschreibung der Elemente ermöglichen. Zusätzlich können XML-Dokumente, wie bereits erwähnt, weitere optionale Bestandteile wie Kommentare und Verarbeitungsanweisungen enthalten, die entweder durch die Zeichenfolge `<!--` und `-->` oder durch `<?` und `?>` eingeschlossen werden. Abbildung 12 stellt ein Beispiel mit den eben erläuterten Elementen dar.

---

78 Vgl. Light (1997), S. 21f.; Priesnitz / Teille (2003), S. 16f.
79 Vgl. Holzner (2002), S. 32.
80 Vgl. Seely (2002), S. 26ff.
81 Vgl. Eddy / Delong (2001), S. 660f.

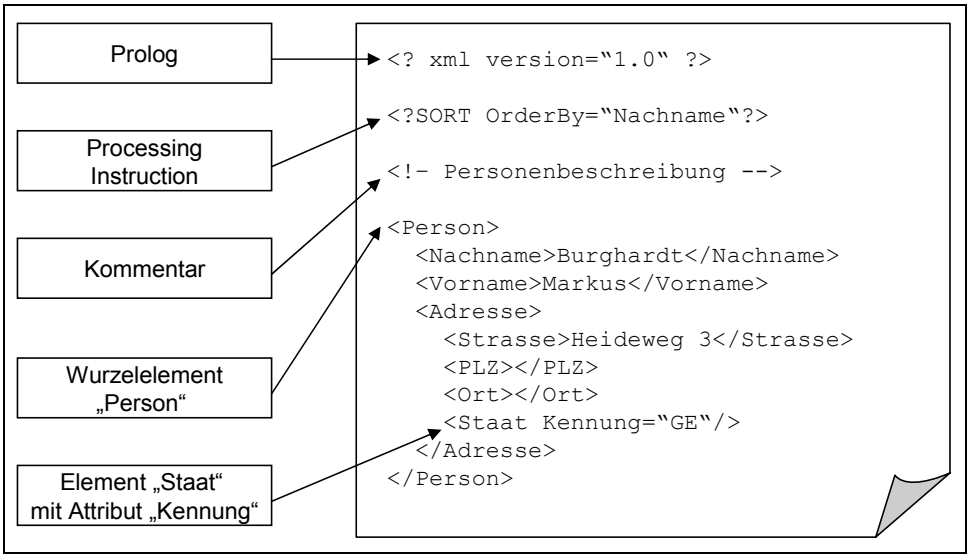

Abbildung 12: Beispiel eines XML-Dokuments

XML-Dokumente, die diese Syntaxregeln einhalten, werden durch die XML-Spezifikation als wohlgeformte XML-Dokumente bezeichnet. Folgt das Dokument darüber hinaus den Strukturbeschreibungen einer zugeordneten Document Type Definition bzw. einem speziellen Schema, wird das XML-Dokument als gültig bezeichnet. Aus dieser Darstellung wird deutlich, dass ein gültiges XML-Dokument immer ein wohlgeformtes XML-Dokument darstellt, die Umkehrung hingegen ist nicht immer gegeben. Der Bezug auf eine Strukturbeschreibung (Document Type Definition / Schema) wird durch einen Verweis im XML-Dokument kenntlich gemacht.

Auch sieht die XML-Spezifikation die Aufteilung der XML-Dokumente auf mehrere Einheiten (engl. Entities) vor, die in verschiedenen physischen Dateien abgelegt werden können. Dadurch wird u. a. eine gewisse Redundanzfreiheit erzeugt, da die Einheiten in verschiedene XML-Dokumente eingebunden werden können.

Strukturbeschreibung

Um die Struktur von XML-Dokumenten zu beschreiben und somit gültige XML-Dokumente zu erstellen, sieht die XML-Spezifikation die beiden Mechanismen der Document Type Definition (DTD) und der XML Schema vor. Prinzipiell werden durch die Strukturbeschreibung die zulässigen Elemente mit den dazugehörigen Attributen und die hierarchische Anordnung der Elemente innerhalb eines XML-Dokuments beschrieben. Die Strukturbeschreibung selbst kann dabei entweder als interne Strukturdefinition ein Bestandteil des XML-Dokuments sein oder aber als externe Strukturdefinition in einer separaten Datei vorliegen. XML-Dokumente, die sich auf eine Strukturbeschreibung

beziehen, werden als Instanzen dieser Strukturdefinition bezeichnet.[82] Um auch Instanzen basierend auf mehreren Strukturdefinitionen zu ermöglichen, wurden XML Namensräume (XML Namespaces) eingeführt. Die beiden Möglichkeiten einer Strukturbeschreibung und die Verwendung von Namensräumen werden nachfolgend dargestellt.

Abbildung 13 zeigt eine DTD. Dabei wird das in Abbildung 4 aufgezeigte XML-Dokument durch diese Strukturdefinition beschrieben. Als Schlüsselwort zur Deklaration von Elementen dient `ELEMENT`, die Deklaration von Attributen geschieht über das Schlüsselwort `ATTLIST`. Mit dem Schlüsselwort `#PCDATA` (Parsed Character Data) werden Elementinhalte als Text deklariert und das Schlüsselwort `EMPTY` zeichnet leere Elemente aus. Man erkennt beispielsweise an der Strukturdefinition, dass das Element Person die Elemente Nachname, Vorname und Adresse enthält und kann daraus auf die hierarchische Struktur schließen.

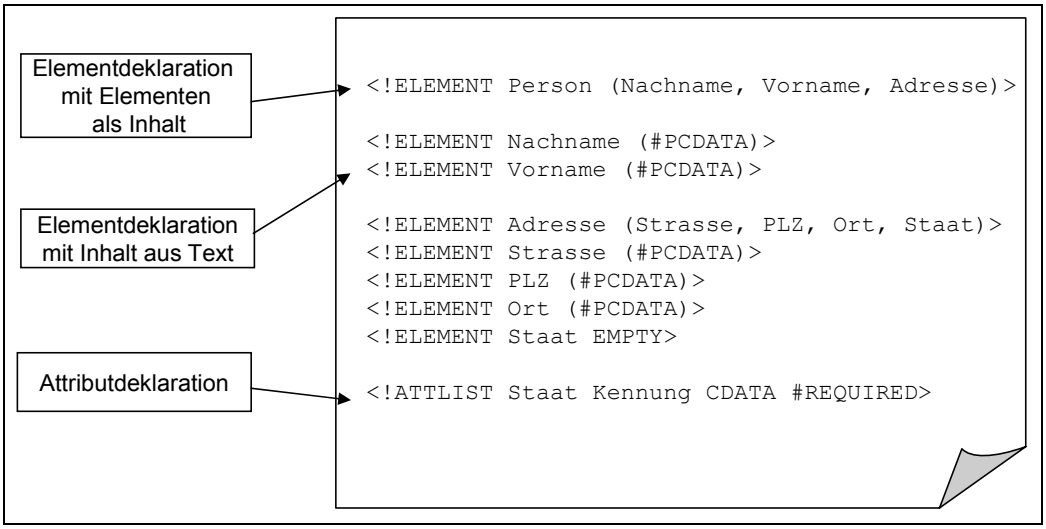

Abbildung 13: Beispiel einer Document Type Definition (DTD)

Auf die gesamten Möglichkeiten der Strukturbeschreibung soll jedoch nicht weiter eingegangen werden, da die Document Type Definition weitere umfangreiche Strukturbeschreibungsmöglichkeiten wie beispielsweise die Definition von optionalen Elementen und Wiederholungsgruppen ermöglicht.

Ein alternativer Spezifikationsmechanismus für die Strukturbeschreibung liegt mit der XML Schema Spezifikation vor, die auf der einen Seite den Beschreibungsumfang von Document Type Definitions erweitert und auf der anderen Seite einige Nachteile der Document Type Definitions aufhebt. [83]

[82] Vgl. Bray / Paoli / Sperberg-McQueen / Maler (2000).
[83] Vgl. Weitzel / Harder / Buxmann (2001), S. 29f.

Folgende wichtige Vorteile sind der XML Schema Spezifikation gegenüber Document Type Definitions anzuführen: [84]

- Schemata sind selbst XML-Dokumente und lassen sich daher mit den gleichen Werkzeugen wie XML-Dokumente bearbeiten.
- Schemata unterstützen Datentypen, die sowohl auf Elementebene als auch auf Attributebene greifen und die möglichen Ausprägungen der Inhalte einschränken.
- Die Kardinalität von Elementen innerhalb von Instanzdokumenten kann anhand von Minimal- und Maximalanzahl genau spezifiziert werden.
- Schemata bieten in Anlehnung an die Objektorientierung sowohl Erweiterungs- (u. a. in Bezug auf die möglichen Datentypen) als auch Vererbungsmechanismen.

Abbildung 14 zeigt ein Schema für das Beispieldokument aus Abbildung 13. Man erkennt deutlich, dass es sich um ein wohlgeformtes XML-Dokument handelt und die Ausprägungen der Inhalte der Elemente explizit durch die Verwendung von Datentypen festgelegt werden können. Auch an dieser Stelle sei aufgrund der Mächtigkeit der XML Schema Spezifikation auf die entsprechenden Quellen verwiesen.[85]

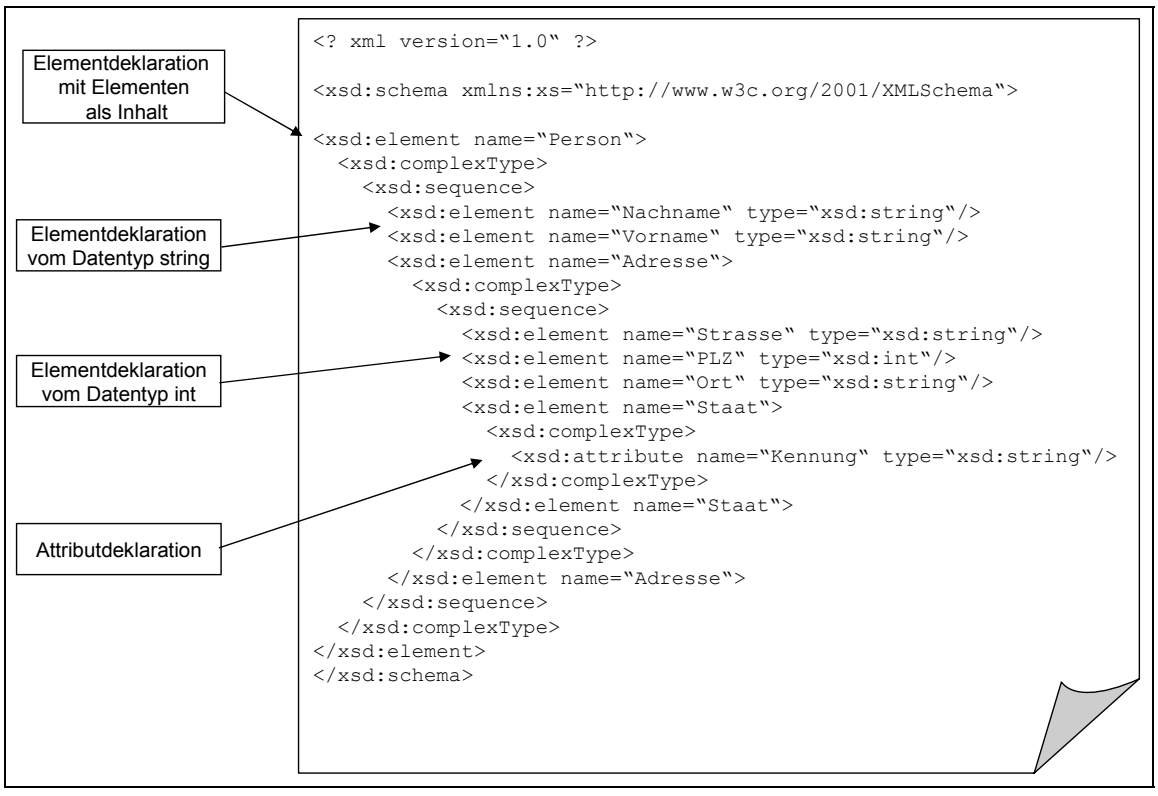

Abbildung 14: Beispiel eines XML Schema

[84] Vgl. Box / Skonnard / Lam (2000), S. 143f.; Hansch / Kuhlins / Schader (2002), S. 363ff.; Mertens (2003), S. 258.

[85] Vgl. zu weiteren Ausführungen bzgl. der XML Schema Spezifikation u. a. Thompson / Beech / Maloney / Mendelsohn (2001).

Die XML Spezifikation sieht prinzipiell vor, dass Instanzen auf Basis genau einer Strukturdefinition möglich sind. Dieses Konstrukt stellt sich allerdings als zu unflexibel heraus.[86] Um die Möglichkeit zu schaffen, XML-Dokumente zu erstellen, die auf mehreren verschiedenen Strukturdefinitionen basieren und somit prinzipiell Instanzdokumente aller genutzten Strukturdefinitionen sind, wurden Namensräume in XML geschaffen. Mit Hilfe dieser Namensräume ist es möglich, Namenskollisionen zu vermeiden, die Entstehen, wenn in mehreren Grammatiken Elemente mit der selben Bezeichnung auftreten und dann keiner Grammatik eineindeutig zugeordnet werden können.[87] Namensräume ordnen somit Elemente mit den dazugehörigen optionalen Attributen über eine Uniform Resource Identifier (URI) einer eindeutigen Strukturdefinition zu[88]. In Abbildung 6 wurden bereits Namensräume verwendet. Die verwendeten Elemente wurden dabei eineindeutig dem Namensraum `xsd` zugeordnet, der mit der entsprechenden XML Schema Strukturdefinition verknüpft ist.

3.3 Dienstkommunikation

Im nachfolgenden Abschnitt wird die Ebene der Dienstkommunikation im Protokollstapel von Web Services näher betrachtet. Dabei werden zuerst mögliche Spezifikationen aufgezeigt, die sich für die Dienstkommunikation bei Web Services eignen und daran anschließend den etablierten Standard auf dieser Ebene und dessen Vorläufer vorzustellen.

3.3.1 Überblick

Im Rahmen der Dienstkommunikation werden durch Web Services die beiden Kommunikationsparadigmen des Remote Procedure Calls und der nachrichtenorientierte Kommunikation umgesetzt. Bei beiden Umsetzungen basiert die Kommunikation bei Web Services auf einem XML-Protokoll. Diese XML-Protokolle lassen sich in zwei Generationen untergliedern. Die erste Generation von XML-Protokollen basiert auf der XML Spezifikation in der Version 1.0. Dazu zählen die Spezifikationen des Web Distributed Data Exchange (WDDX) als auch die Spezifikation von XML-RPC, einem Mechanismus zur Spezifikation von RPC's auf Basis der Metasprache XML. Den XML-Protokollen der zweiten Generationen gehören das Simple Object Access Protocol (SOAP) als auch die Direct Internet Message Encapsulation (DIME) an. Dabei verwenden XML-Protokolle der zweiten Generation im

[86] Vgl. Michel (1999), S. 147ff.
[87] Vgl. Bray / Hollander / Layman (1999).
[88] Vgl. Mertens (2003), S. 258f.

Gegensatz zu Protokollen der ersten Generation sowohl XML Namensräume als auch XML Schemata zur Datentypdefinition.[89] Dabei haben sich in der Praxis neben XML-RPC auch das Simple Object Access Protocol (SOAP) bei Web Services etabliert. SOAP stellt eine Weiterentwicklung von XML-RPC dar und vereint alle Vorteile von XML-RPC. Aus diesem Grund liegt der Schwerpunkt der nachfolgenden Betrachtung auf dem Simple Object Access Protocol. Jedoch soll auch eine kurze Einführung in XML-RPC gegeben werden.

3.3.2 XML-RPC

XML-RPC ist ein relatives einfaches, plattform- und programmiersprachenunabhängiges Protokoll für den Aufruf entfernter Methoden oder Prozeduren über ein Netzwerk. Dabei wird als Nachrichtensprache XML und als Transportprotokoll HTTP genutzt. Sowohl der Aufruf einer Methode und die dafür benötigten Informationen wie der Methodenname und die Parameter als auch die Antwort der Methode werden als wohlgeformte XML-Dokumente.[90]
In der Spezifikation von XML-RPC werden neben dem Aufbau der Anfrage- und Antwortnachrichten auch Datentypen definiert, die innerhalb der Nachrichten zur Anwendung kommen können. Dabei werden neben einfachen Datentypen (string, int, etc.) auch komplexe Strukturen wie beispielsweise Felder spezifiziert.[91] Abbildung 15 verdeutlich den Aufbau einer Anfragenachricht auf Basis von XML-RPC und versieht die genutzten Elemente mit deren Bedeutung.

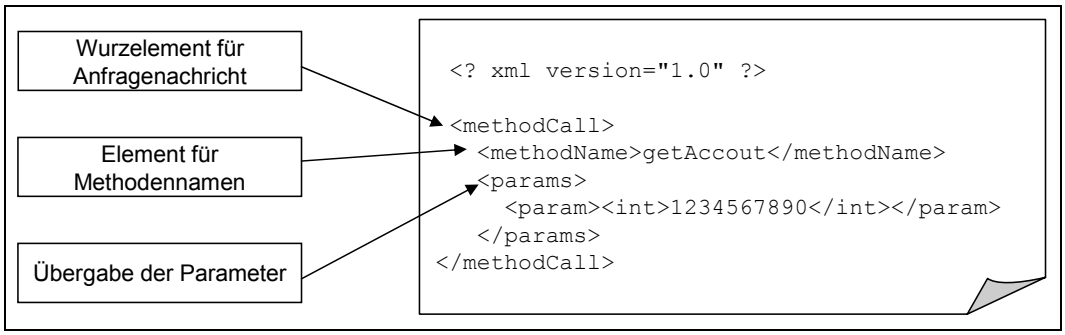

Abbildung 15: Anfragenachricht auf Basis von XML-RPC

Die Spezifikation von XML-RPC erfolgte bereits 1998 durch die Firma UserLand Software und somit unabhängig von der Entwicklung von Web Services. Jedoch erwies sich das Protokoll XML-RPC als sehr unflexibel, da keine eigenen XML-Strukturen übertragen werden konnten und somit ein nicht unerheblicher Transformationsaufwand anfiel. Darüber hinaus ist keine strukturelle Erweiterbarkeit von XML-RPC möglich. Daher wurde XML-RPC in

[89] Vgl. Graham / Simeonov / Boubez (2002), S. 33ff.
[90] Vgl. Saint Laurent / Johnston / Dumbill (2001), S. 13ff.
[91] Vgl. Cerami (2002), S. 31ff.

Zusammenarbeit mit Microsoft weiterentwickelt und als Simple Object Access Protocol (SOAP) verabschiedet.[92]

Im Umfeld der Web Services Innovation wird XML-RPC als mögliches Protokoll für die Dienstkommunikation heutzutage nur sehr wenig Aufmerksamkeit geschenkt. Jedoch eignet sich das Protokoll für einfache Web Services besonders gut, da es aufgrund der einfachen Spezifikation in Bezug auf das zugrunde liegende XML-Vokabular mit relativ wenig Aufwand erlernbar ist. Demgegenüber ist der Nachteil im Bezug auf die Flexibilität der übertragenen Nachrichten und die fehlende Erweiterbarkeit nicht zu unterschätzen.[93]

3.3.3 Simple Object Access Protocol (SOAP)

Das Simple Object Access Protocol (SOAP) stellt eine Weiterentwicklung der Spezifikation von XML-RPC dar, um die aufgezeigten Problematik der fehlenden Flexibilität in Bezug auf die zu übertragenden Inhalte als auch die Erweiterbarkeit hinsichtlich der vorgegebenen Struktur zu lösen.

SOAP wurde ab 1998 unter Zusammenwirken der Firmen Microsoft, DevelopMentor, Lotus, IBM und SAP entwickelt und schließlich in der Spezifikation 1.1 im Mai 2000 vorgestellt. Die Verantwortung für die Weiterentwicklung der Spezifikation liegt heutzutage beim World Wide Web Consortium (W3C), das im Juni 2003 die Spezifikation 1.2 vorstellte. Diese Spezifikation umfasst neben einer Einführung die beiden Hauptabschnitte des SOAP Messaging Frameworks und des SOAP Adjuncts. Dabei wurde die Einfachheit als auch die Erweiterbarkeit als Zielsetzung für das Protokoll angestrebt.[94]

SOAP selbst ist ein XML-basiertes und dadurch einfaches, sprach-, plattformunabhängiges Kommunikationsprotokoll zum Austausch strukturierter Informationen. Als Einsatzgebiete lassen sich die Entwicklung verteilter Anwendung im Allgemeinen als auch Web Services im Speziellen ausmachen. Im Bereich der Web Services hat sich SOAP als Standard für die Nachrichtenkapselung etabliert. Hinsichtlich der bereits aufgeführten Kommunikations-paradigmen des Remote Procedure Calls (RPC) als auch der nachrichtenorientierten Kommunikation werden durch SOAP keine Einschränkungen vorgenommen. SOAP ermöglicht somit die Kommunikation anhand beider Kommunikationsparadigmen. Die nachrichtenorientierte Kommunikation wird durch die Möglichkeit geschaffen, beliebige auf

---

92 Vgl. Rawolle / Burghardt (2002), S. 41.
93 Vgl. Saint Laurent / Johnston / Dumbill (2001), S 169ff.; Krüger / Deutschmann (2002), S. 258ff.
94 Vgl. Box / Ehnebuske / Kakivaya / Layman / Mendelsohn (2002); Mitra (2003); Gudgin / Hadley / Mendelsohn / Moreau / Nielsen (2003a); Gudgin / Hadley / Mendelsohn / Moreau / Nielsen (2003b).

Basis von XML kodierte Informationen unter Anwendung von SOAP zu übermitteln. Darauf aufbauend werden spezielle Rahmenbedingungen für die Abbildung eines Remote Procedure Calls vorgegeben, die die Struktur der Anfrage- und Antwortnachricht als auch der Fehlernachricht regeln.[95]

Aufbau einer SOAP-Nachricht

SOAP selbst definiert somit ein Rahmenwerk zur Beschreibung von Nachrichten und deren inhaltlichen Verarbeitung. Dabei stellen SOAP Nachrichten immer wohlgeformte XML-Dokumente dar. Die SOAP Nachricht besteht aus einem Umschlag, dem SOAP Envelope, der als Wurzelement der Nachricht dient. Der Umschlag kapselt den optionalen Kopf (`SOAP:Header`) und einen obligatorischen Körper (`SOAP:Body`) der Nachricht. Der Kopf selbst enthält anwendungsspezifische Informationen wie beispielsweise Angaben zur Transaktionssteuerung oder zum Kontext. Auch können im Kopf Verarbeitungsregeln hinterlegt werden. Da eine Nachricht auf dem Weg vom Sender zum Empfänger mehrere Zwischenstationen (Intermediäre) durchlaufen kann, sind mehrere individuelle Kopfinhalte möglich. Im Hauptteil werden die Anwendungs- bzw. Nutzdaten der Nachrichten, der so genannte Payload, transportiert. Im Falle der nachrichtenorientierten Kommunikation ist dort das zu übertragene XML-Dokument angesiedelt, im Falle eines Remote Procedure Calls sind dort alle notwendigen Informationen wie der Methodenname und die Parameter im Fall des Methodenaufrufs beziehungsweise die Rückgabewerte oder Fehlermeldungen im Falle der Antwortnachricht eingebettet.[96] Dabei werden alle Elemente, aus denen sich die Nachricht zusammensetzt, also auch der Umschlag, der Kopf und der Körper eineindeutig unter Verwendung von XML Namensräumen spezifiziert.[97]

Abbildung 16 zeigt die eben dargestellte Struktur einer Nachricht abstrakt.

---

[95] Vgl. Krüger / Deutschmann (2002), S. 259ff.; Snell / Tidwell / Kulchenko (2002), S. 12.; Curbera / Duftler / Khalaf / Nagy / Mukhi / Weerawarana (2002), S. 86ff.

[96] Vgl. Langner (2003), S. 127.; Keidl / Kemper / Seltzsam / Stocker (2002), S. 300ff.; Box / Ehnebuske / Kakivaya / Layman / Mendelsohn (2002).

[97] Vgl. Basha (2002), S. 45ff.; Mitra (2003).

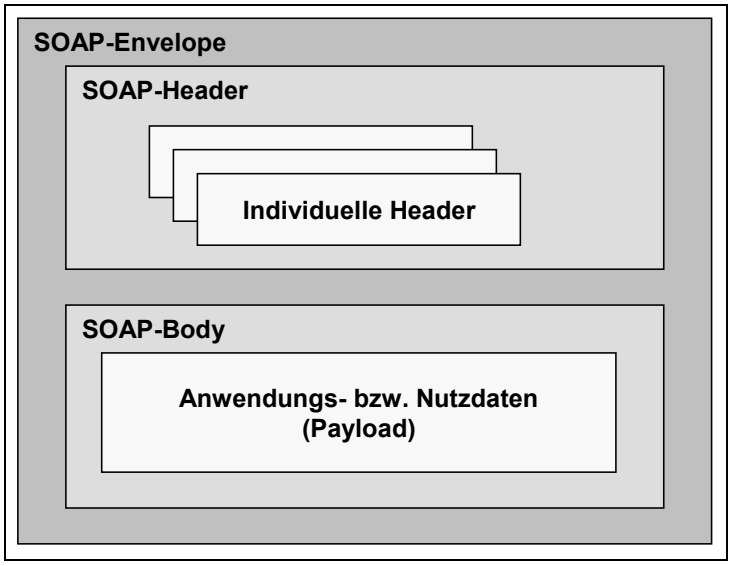

Abbildung 16: Abstrakte Struktur einer SOAP-Nachricht

Darüber hinaus werden in einem Datenmodell innerhalb der SOAP Spezifikation analog zu XML-RPC Datentypen und Datenstrukturen für die Nachrichten festgelegt. Innerhalb des Datenmodells werden skalare Datentypen (string, int, etc.) von zusammengesetzten Datentypen (array, structs, etc.) unterschieden. Jedoch können auch eigene Datentypen innerhalb der Nachricht übertragen werden, wodurch die Erweiterbarkeit gewährleistet wird.[98]

Abbildung 17 visualisiert die abstrakte Struktur einer SOAP-Nachricht an einem konkreten Beispiel. Dabei werden auch die angesprochene Zuordnung von Namensräumen zu den einzelnen Elementen innerhalb der Nachricht sowie die zuvor diskutierte Festlegung von Datentypen für die übertragenen Parameter deutlich.

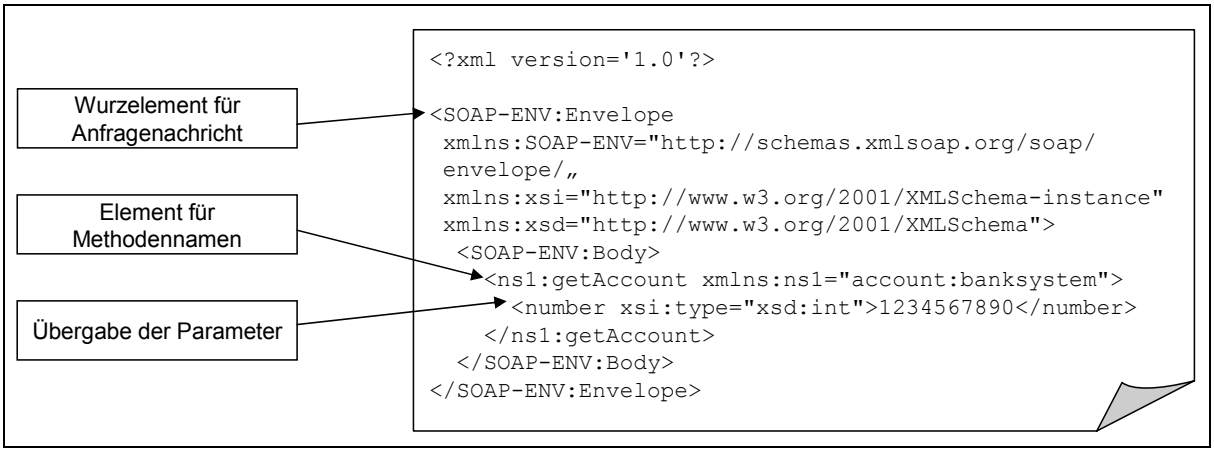

Abbildung 17: Anfragenachricht auf Basis von SOAP

Um auch die Flexibilität in Bezug auf das genutzte Transportprotokoll zu erhalten, sieht SOAP keine feste Bindung an das Transportprotokoll vor. Es sind somit eine Vielzahl von Transportprotokollen wie HTTP, RMI/IIOP, SMTP, FTP und MQ oder sogar JMS möglich.

[98] Vgl. Scribner / Stiver (2000); Gudgin / Hadley / Mendelsohn / Moreau / Nielsen (2003b).

Allerdings sind in der Spezifikation zur Zeit nur konkrete Bindungen an die Transportprotokolle HTTP und SMTP hinterlegt.[99]

SOAP zeichnet sich somit wie bereits besprochen durch seine Simplizität und Erweiterbarkeit aus, da es auf der Metasprache XML basiert. Darüber hinaus wird eine hohe Flexibilität in Hinblick auf die Struktur der zu übertragenen Informationen, das genutzte Transportprotokoll sowie auf die möglichen Kommunikationsmodelle durch die Spezifikation erreicht. Somit ist SOAP programmiersprachen- und plattformunabhängig einsetzbar. Allerdings werden in der Spezifikation von SOAP weiterführende Aspekte wie beispielsweise die Sicherheit der Übertragung nicht behandelt.[100]

Ablauf der Kommunikation

Beim Ablauf eines Kommunikationsvorgangs auf Basis von SOAP erstellt ein so genannter SOAP Client aus allen von der Anwendung bereitgestellten Informationen eine SOAP Nachricht, die mittels eines Transportprotokolls übertragen wird. Auf der Serverseite wird die Nachricht durch einen so genannten SOAP Server zerlegt und die Informationen an die Zielanwendung weitergegeben. Das Ergebnis der Verarbeitung durchläuft den Kommunikationsablauf in umgekehrter Richtung. Auf der Ebene des Transportprotokolls sind auch die angesprochenen Zwischenstationen (Intermediäre) der Nachrichtenübermittlung angesiedelt, die eine Art Vermittlungsaufgabe zwischen dem Sender und dem Empfänger übernehmen. Entweder überwachen die Intermediäre die komplette Kommunikation, also sowohl die Anfrage als auch die Antwort (Intermediärtyp A), oder sie treten nur bei der Anfrage (Intermediärtyp B) oder der Antwort (Intermediärtyp C) in Erscheinung. Es sind somit drei Arten von Intermediären denkbar. Auch können mehrere Intermediäre unterschiedlicher Typen hintereinander zwischen dem Sender und dem Empfänger der Nachricht angesiedelt sein.[101]

Abbildung 18 veranschaulicht diesen Ablauf sowohl für die Kombination SOAP und HTTP als Transportprotokoll sowie eines Web Services als Zielanwendung als auch die drei Arten möglicher Intermediäre.

---

[99] Vgl. Burghardt / Gehrke / Schumann (2003a), S. 72ff.; Snell / Tidwell / Kulchenko (2002), S. 33f.; Box / Ehnebuske / Kakivaya / Layman / Mendelsohn (2002), S. 49.; Box / Ehnebuske / Kakivaya / Layman / Mendelsohn (2002).

[100] Vgl. Scribner / Stiver (2000), S. 77f.;

[101] Vgl. Rieck / Dostal / Schandl / Sieb (2003), S. 297f.

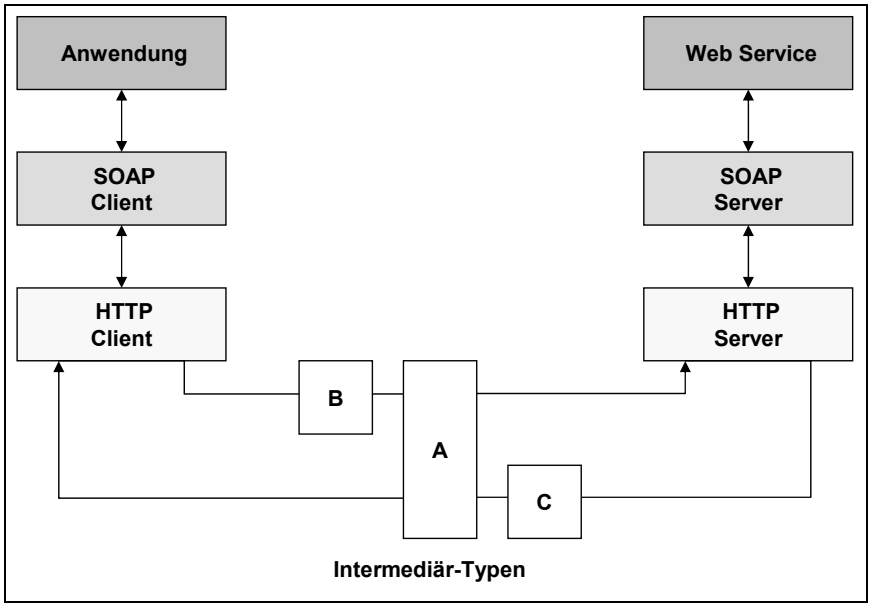

Abbildung 18: Ablauf einer SOAP/HTTP – Kommunikation eines Web Service

3.4 Dienstbeschreibung

Im nachfolgenden Abschnitt wird die Ebene der Dienstbeschreibung im Protokollstapel von Web Services näher betrachtet. Dabei werden zuerst mögliche Spezifikationen aufgezeigt, die sich für die Beschreibung der Schnittstellen und somit der Funktionalitäten bei Web Services eignen und daran anschließend den etablierten Standard auf dieser Ebene ausführlich erläutert.

3.4.1 Überblick

Die Dienstbeschreibung ermöglicht es, Informationen über die Funktionalität und die angebotenen Schnittstellen eines Web Services bereitzustellen. Die Aufgabe der Dienstbeschreibung übernimmt der Dienstanbieter. Wird diese Dienstbeschreibung öffentlich zugänglich hinterlegt bzw. einem Dienstnachfrager übermittelt, so ist dieser in der Lage, anhand dieser Informationen einen Web Service in eine Anwendung einzubinden und diesen zu nutzen. In der Literatur werden funktionale und nichtfunktionale Beschreibungen spezifiziert.[102]

Innerhalb der funktionalen Beschreibung wird zwischen der Implementierung und dem Interface unterschieden. Dabei wird im Interface die abstrakte Beschreibung des Web

[102] Vgl. Graham / Simeonov / Boubez (2002), S. 41ff.

Services niedergelegt, wodurch eine Wiederverwendbarkeit des Interfaces erreicht wird. Innerhalb des Interface werden neben dem Aufbau der Anfrage- und Antwortnachricht selbst, die der Web Service erwartet, auch das genutzte Transportprotokoll hinterlegt. Die Implementierung beschreibt dann die konkreten Details der Umsetzung durch den Anbieter. Dort wird dann der Verbindungsendpunkt, also die Netzwerkadresse, hinterlegt, über welche der Dienst durch den Dienstnutzer ansprechbar ist. Auf Basis dieser funktionalen Beschreibung kann der Dienstnutzer den Dienst in eine Anwendung einbinden und nutzen.

Die nichtfunktionale Beschreibung beinhaltet Informationen, die über die funktionale Beschreibung hinausgehen, und somit nicht direkt für die Einbindung und Nutzung des Web Services nötig sind. Dazu zählen beispielsweise Informationen über den Dienstanbieter, aber auch Informationen, die zusätzlich vor der Nutzung des Web Services zwischen dem Dienstanbieter und Dienstnutzer ausgetauscht werden müssen.

Im diese Dienstbeschreibung von Web Services vornehmen zu können, können unterschiedliche Spezifikationen angewendet werden. Dazu zählt beispielsweise das Resource Description Framework (RDF)[103] oder die DARPA Agent Markup Language Ontology for Web Services (DAML-S)[104]. Jedoch hat sich in Bereich der Dienstbeschreibung die Web Service Description Language (WSDL)[105] etabliert, die nachfolgend dargestellt werden soll.

3.4.2 Web Service Description Language (WSDL)

Die Web Service Description Language (WSDL) legt fest, in welcher Weise die für den Aufruf eines Web Services notwendigen Informationen in einem XML-Dokument niedergelegt werden müssen. Dabei wurde WSDL durch Ariba, Microsoft und IBM entwickelt und eine Standardisierung durch das W3C erreicht. WSDL liegt zur Zeit in Version 1.2 vor (Stand Oktober 2003). Die in dieser Form ablegte Dienstbeschreibung kann dann aufgrund der Nutzung von XML als Basis plattform- und programmiersprachenunabhängig eingesetzt werden.[106]

WSDL selbst betrachtet einen Web Services als eine Menge von Verbindungsendpunkten in einem Netzwerk, die in der Lage sind, Nachrichten auf Basis von XML zu empfangen, diese

[103] Vgl. Klyne / Carroll (2003).
[104] Vgl. Martin / Burstein / Lassila / Paolucci / Payne / McIlraith (2002).
[105] Vgl. Christensen / Curbera / Meredith / Weerawarana (2002).
[106] Vgl. Keidl / Kemper / Seltzsam / Stocker (2002), S. 320ff.; Christensen / Curbera / Meredith / Weerawarana (2002); Chinnici / Gudgin / Moreau / Weerawarana (2003).

zu verarbeiten und optional Antwortnachrichten zu generieren.[107] Dabei können die Nachrichten entweder XML-Dokumente oder entfernte Methodenaufrufe enthalten. Innerhalb der Beschreibung wird festgelegt, wie die Nachrichten strukturell beschaffen sein müssen sowie das für die Übertragung genutzte Transportprotokoll und der Port. Es erfolgt durch WSDL somit eine Beschreibung unabhängig von der gewählten Implementierung, wodurch die Änderung der Implementierung keinen Einfluss auf die Dienstbeschreibung nach sich zieht, es sei denn, die Funktionalität bzw. die Signatur des Dienstes ändert sich.

WSDL kapselt also alle Informationen, die zur Nutzung eines Web Services nötig sind, es handelt sich somit um eine Art Interface Definition Language (IDL) für Web Services auf Basis von XML.[108]

Aufbau eines WSDL-Dokuments

Abbildung 19 veranschaulicht schematisch den Aufbau eines WSDL-Dokuments, auf dessen einzelne Elemente nun kurz eingegangen wird. Die Verbindungen zeigen dabei die Referenzierung zwischen den Elementen auf.[109]

[107] Vgl. Chappell / Jewell / Dalheimer (2003), S. 80.
[108] Vgl. Langner (2003), S. 168ff.; Chappell / Jewell / Dalheimer (2003), S. 80ff.
[109] Vgl. Keidl / Kemper / Seltzsam / Stocker (2002), S. 320f.

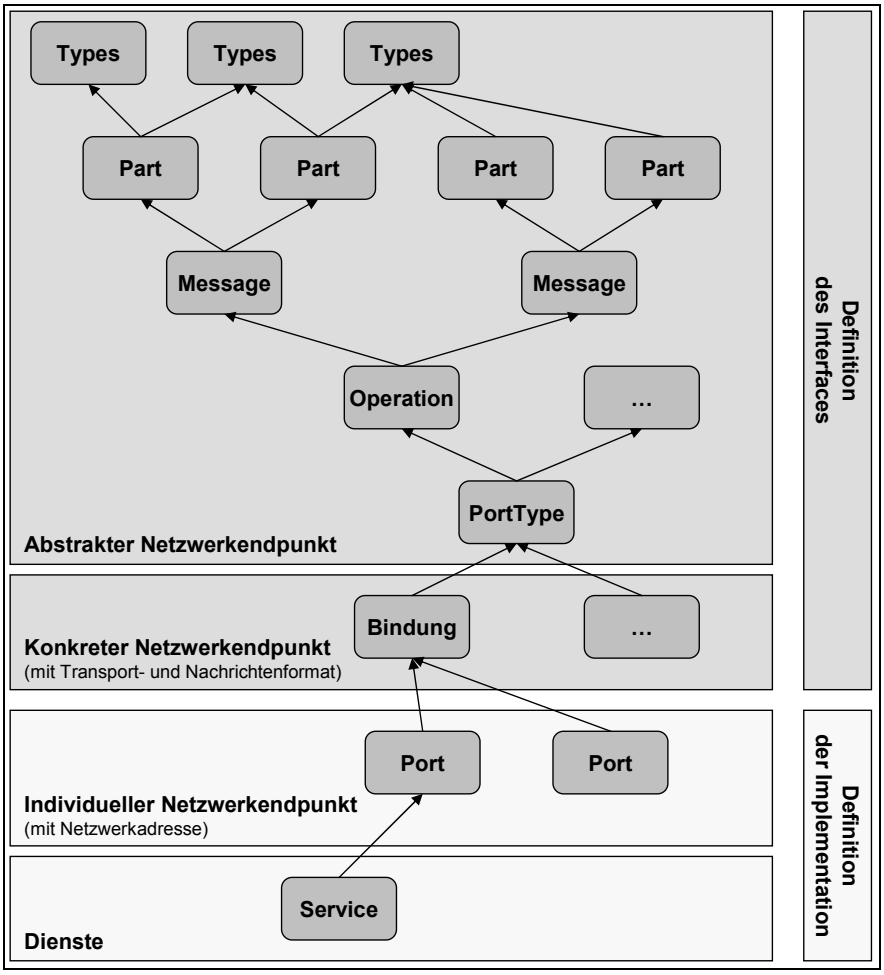

Abbildung 19: Aufbau eines WSDL-Dokuments

Ein Dienstbeschreibung in WSDL kann in die Teile der Definition von Interfaces (Schnittstellenbeschreibung) und der Definition der Implementation (Implementierungsbeschreibung) eingeteilt werden. Dabei erfolgt die Definition von Interfaces in einer abstrakten Form, die es gestattet, bei verschiedenen Umsetzungen eine Wiederverwendbarkeit der Interfacedefinition zu erzielen. Die Definition der Implementation übernimmt dann die konkrete Zuordnung des Web Services zu einer Netzwerkadresse.

Innerhalb der Definition des Interfaces werden ein Datentyp-, ein Nachrichten-, ein Porttyp- sowie ein Anbindungselement als Hauptelemente spezifiziert, die nachfolgend erläutert werden sollen. Alle Elemente werden innerhalb der Dienstbeschreibung durch einen eindeutigen Namen gekennzeichnet, der zur Referenzierung innerhalb der Dienstbeschreibung genutzt wird.[110]

- Das Datentypelement (`<wsdl:types>`) dient zur Definition von eigenen Datentypen, die nicht in der XML Schema Spezifikation vorgesehen sind und die innerhalb

[110] Vgl. Keidl / Kemper / Seltzsam / Stocker (2002), S. 320ff.; Chappell / Jewell / Dalheimer (2003), S. 82ff.; Newcomer (2002), S. 65ff.; Christensen / Curbera / Meredith / Weerawarana (2002); Chinnici / Gudgin / Moreau / Weerawarana (2003).

der ausgetauschten Nachrichten verwendet werden sollen. Dabei sollten alle eigenen Datentypen ebenfalls aus Gründen der Interoperabilität unter Nutzung von XML Schema spezifiziert werden.

- Das Nachrichtenelement (`<wsdl:message>`) dient zum Festlegen der Anfrage- und Antwort- sowie der Fehlernachrichten, die ein Web Service verarbeitet und die somit zwischen dem Dienstanbieter und dem Dienstnachfrager ausgetauscht werden. Dabei besteht jede Nachricht aus logischen Bestandteilen (`<wsdl:part>`), die eine nähere Spezifizierung der Nachricht vornehmen, indem beispielsweise über Attribute die genutzten Datentypen zugeordnet werden. Dabei sind die Bestandteile selber Kindelemente der Nachricht.
- Das Porttypelement (`<wsdl:portType>`) kapselt eine Menge von Operationen, die dem Dienstnutzer zur Verfügung stehen. Die Operationen (`<wsdl:operation>`) besitzen Kindelemente, über die die Verweise auf die beteiligten Anfrage-, Antwort- und Fehlernachrichten abgebildet werden. An dieser Stelle wird auch der Methodenname festgelegt, den der Dienstnutzer zur Aktivierung des Dienstes ansprechen muss.
- Das Anbindungselement (`<wsdl:bindung>`) spezifiziert das genutzte Kommunikationsparadigma sowie das Transportprotokoll, welches zur Übermittlung der Nachrichten an den Verbindungsendpunkt genutzt werden soll. Es wird somit die Kommunikation zwischen Dienstanbieter und Dienstnutzer definiert.

Neben dieser abstrakten Definition des Interfaces wird bei der Definition der Implementation nun die konkreten Details der Umsetzung spezifiziert. Dabei existieren das Portelement und das Serviceelement als Hauptelemente. Auch diese sollen nachfolgend erläutert werden:

- Das Portelement (`<wsdl:port>`) spezifiziert die möglichen Verbindungsendpunkte eines Web Services und somit die Netzwerkadresse, über den die Nutzung erfolgen kann. Es erfolgt auch eine Referenzierung auf ein Anbindungselement. Hier wird der Kreis zwischen der abstrakten und der konkreten Definition geschlossen. Das Portelement wird über einen Namen eindeutig im WSDL-Dokument gekennzeichnet.
- Das Serviceelement (`<wsdl:service>`) identifiziert den Web Services und enthält die Portelemente als Kindelemente. Das Serviceelement bündelt somit eine Sammlung von Kommunikationsendpunkten, die den Web Services implementieren, indem als Kindelemente Referenzen auf die Portelemente hinterlegt werden.

Optional kann ein Dokumentationselement (`<wsdl:documentation>`) vorhanden sein, welches eine allgemeine Beschreibung des Dienstes ermöglicht. Dieses Dokumentations-

element ist die einzige Möglichkeit, eine nicht funktionale Beschreibung des Web Services vorzunehmen. Auch ist eine automatische Weiterverarbeitung der Inhalte des Dokumentationselements aufgrund der fehlenden Standardisierung nicht gegeben.

Auf Basis dieser Spezifizierung des Dienstes ist es nun möglich, den Dienst einzubinden und zu nutzen. Abbildung 20 verdeutlicht den in Abbildung 18 aufgezeigten abstrakten Aufbau eines WSDL-Dokuments an einem Beispiel. Dabei wurde das bereits in Abschnitt der Dienstkommunikation eingeführte Beispiel weiterverwendet und für diesen Dienst eine Dienstbeschreibung niedergelegt.

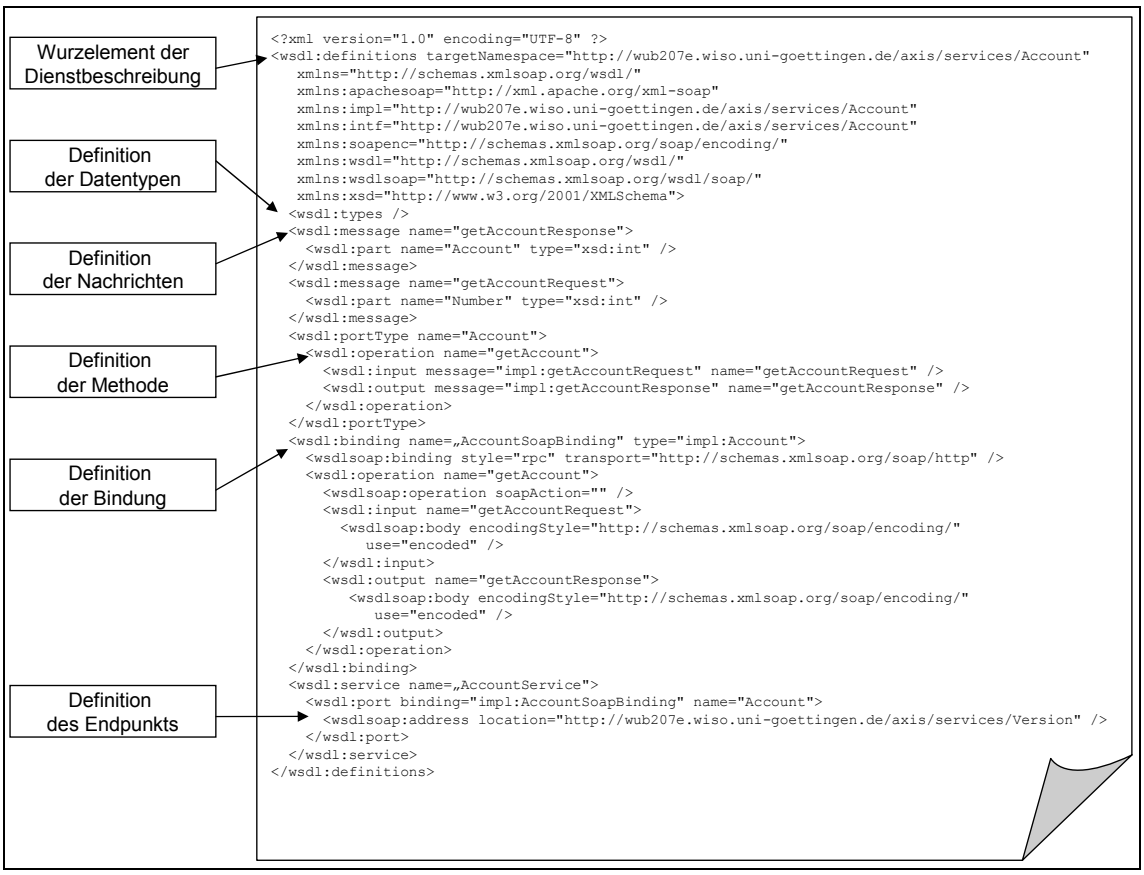

Abbildung 20: Beispiel einer Dienstbeschreibung in WSDL

3.5 Dienstverzeichnis – UDDI

Im nachfolgenden Abschnitt wird die Ebene des Dienstverzeichnisses im Protokollstapel von Web Services näher betrachtet. Dabei werden zuerst mögliche Arten von Dienstverzeichnissen und Spezifikationen aufgezeigt, die sich für die Bereitstellung eines Dienstverzeichnisses eignen. Daran anschließend wird das etablierteste Dienstverzeichnis auf dieser Ebene ausführlich erläutert.

3.5.1 Überblick

Um den Dienstnutzer die Möglichkeit zu geben, einen Dienst eines Dienstanbieters zu aktivieren und diesen in eine Anwendung einzubinden, muss der Dienstnutzer über die Existenz des Dienstes in geeigneter Weise informiert werden. Wie bereits dargestellt, untergliedert sich diese Informationsphase in die beiden Unterabschnitte der Dienstveröffentlichung und der Dienstsuche. Die Dienstveröffentlichung kann dabei entweder statisch oder dynamisch Erfolgen. Die Phase der Dienstsuche fällt jedoch nur bei der dynamischen Veröffentlichung der Dienstbeschreibung an.[111]

Im Rahmen der statischen Dienstveröffentlichung kann diese Übermittlung der zur Dienstbeschreibung in Form einer E-Mail erfolgen. Auch ist eine Ablage der Dienstbeschreibung in einer Datei denkbar, die dann über einen FTP-Server zum Download zur Verfügung gestellt wird. Der Dienstnutzer enthält bei der direkten Veröffentlichung somit immer eine Kopie der Dienstbeschreibung.[112]

Bei der dynamischen Dienstveröffentlichung wird die Dienstbeschreibung bzw. ein Verweis auf diese in einem Verzeichnis hinterlegt, welches dann durch den potentiellen Dienstnutzer durchsucht werden kann. Dabei unterschieden sich diese Verzeichnis auf der einen Seite hinsichtlich ihres Funktionsumfangs und auf der anderen Seite hinsichtlich des Umfangs der abgelegten Dienstbeschreibungen.

Die einfachste Form von Verzeichnissen stellen Webseiten dar, die eine Auflistung verfügbarer Web Services anbieten und eine Referenz auf die Dienstbeschreibung vorhalten. Die Web Services werden in diesen einfachen Verzeichnissen in der Regel durch Nutzung von HTML-Formularen veröffentlicht. Eine Suchmöglichkeit auf den Webseiten ist meistens nicht vorhanden. Die Webseite von XMethods[113] kann als ein solch einfaches Verzeichnis angesehen werden. Darüber hinaus spezifizieren die Ansätze des Advertisement and Discovery of Services (ADS) von IBM[114] als auch der Ansatz des Discovery of Web Services von Microsoft[115] einfache Mechanismen auf Basis von HTTP-GET, die den Bezug von Dienstbeschreibungen von einer vorgegebenen URL erlauben.

Wird statt der Referenz auf die Dienstbeschreibung auch die Möglichkeit geboten, eine komplette Dienstbeschreibung im Verzeichnis abzulegen, so wird von einem WSDL-Repository gesprochen. Diese erweiterten Verzeichnisse können auch über erweitere

---

[111] Vgl. Beimborn / Mintert / Weitzel (2002), S. 277f.; Bettag (2001), S. 302ff.
[112] Vgl. Kreger (2001), S. 21.
[113] Vgl. http://www.xmethods.com
[114] Vgl. Nagy / Curbera / Weerawaranna (2000).
[115] Vgl. Skonnard (2002), S. 23ff.

Mechanismen zur Veröffentlichung und Funktionalitäten zur Suche veröffentlichter Web Services verfügen.

Eines dieser erweiterten Verzeichnisse mit den zur Zeit umfangreichsten Funktionalitäten zur Veröffentlichung und zur Suche ist ein Verzeichnis mit dem Namen Universal Description, Discovery and Integration (UDDI).[116] Nachfolgend wird somit UDDI als Verzeichnisdienst betrachtet, da er sich im Umfeld der Web Service Innovation etabliert hat.

Abbildung 21 stellt die möglichen Veröffentlichungsformen und die dafür existierenden Möglichkeiten in einer Übersicht dar. Dabei wird anhand der angebotenen Funktionalität eine Aufsplittung der Möglichkeiten bei der dynamischen Veröffentlichung erreicht.[117]

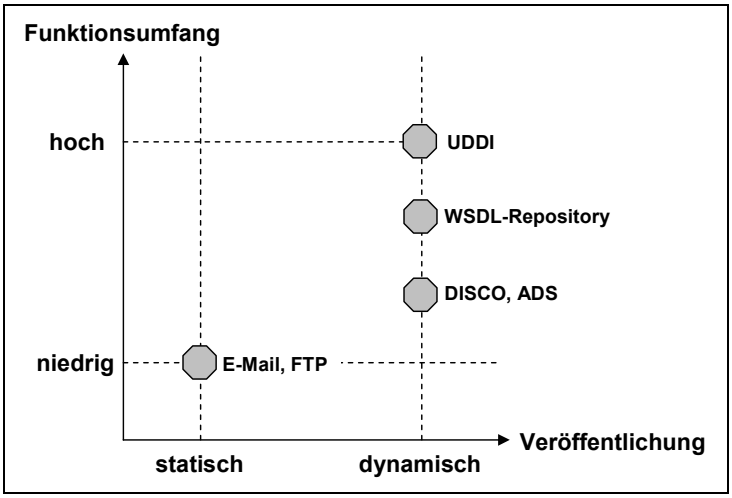

Abbildung 21: Veröffentlichungsmöglichkeiten von Web Services

3.5.2 Universal Description, Discovery and Integration (UDDI)

Die Grundidee von Universal Description, Discovery and Integration (UDDI) ist es, ein Verzeichnis zur Verfügung zu stellen, welches zur Veröffentlichung und zur Suche von Web Services dient. Die erste Spezifikation in Version 1.0 von UDDI wurde bereits im September 2000 durch die Organisation UDDI.org entwickelt, in der über 200 Softwareunternehmen vertreten sind. Die Weiterentwicklung dieser Basisspezifikation erfolgte durch das Standardisierungsgremium OASIS. Mittlerweile wurde die Spezifikation in Version 3.0 im Juli 2002 verabschiedet.[118]

---

[116] Vgl. Bellwood / Clément / Ehnebuske / Hately / Hondo / Husband / Januszewski / Lee / Munter / von Riegen (2002).
[117] Vgl. Kreger (2001), S. 20ff.; Badach / Rieger / Schmauch (2003), S. 338.
[118] Vgl. Bellwood / Clément / Ehnebuske / Hately / Hondo / Husband / Januszewski / Lee / Munter / von Riegen (2002).

UDDI selbst stellt aus konzeptioneller Sicht ein verteiltes Datenbanksystem zur Speicherung der Dienstbeschreibung und zusätzlicher Metadaten der Dienstanbieter dar. Dafür werden innerhalb der UDDI Spezifikation neben den zu erfassten Daten ein einheitliches Datenschema für die Speicherung der Daten definiert. Darüber hinaus spezifiziert UDDI die Zugriffsmöglichkeiten auf das Datenbanksystem und eine Replikationsstrategie.[119]

In der Spezifikation in Version 3.0 unterscheidet UDDI mit privaten, halbprivaten und öffentlichen Verzeichnissen drei unterschiedliche Verzeichnisarten. Private Verzeichnisse befinden sich innerhalb eines Intranets eines Unternehmens und können beispielsweise als Testumgebung verwendet werden. Halbprivate Verzeichnisse sind ebenfalls innerhalb eines Unternehmens angesiedelt, jedoch treten neben den Anwendern innerhalb des Unternehmens ebenfalls Anwender von Geschäftspartner in Erscheinung. Innerhalb dieser Verzeichnisse können beispielsweise Dienste veröffentlicht werden, die durch die Geschäftspartner genutzt werden können. Die öffentlichen Verzeichnisse erfahren keine Einschränkung der Nutzer und werden gegenwärtig von IBM, Microsoft, NTT Communications und SAP betrieben.[120]

Datenumfang und Datenstruktur

Der Verzeichnisdienst UDDI unterscheidet drei verschiedene Klassen von Informationen, die gespeichert werden:[121]

- Weiße Seiten: Die Klasse von Informationen umfasst allgemeine Informationen über den Dienstanbieter. Dazu zählen Angaben über das Unternehmen, wie der Firmenname, die Adresse, die Telefonnummer und Daten über die Kontaktperson innerhalb des Unternehmens.
- Gelbe Seiten: Diese Klasse von Informationen ermöglicht später die komfortable Suche innerhalb der im UDDI Verzeichnis abgelegten Daten. Hier findet eine Kategorisierung der Dienstanbieter und des Dienstes statt, wobei eine vorgegebene Taxonomie verwendet wird, die aber durch den Dienstanbieter beliebig verfeinert werden kann. So ist beispielsweise beim Dienstanbieter eine Einordnung sowohl

[119] Vgl. Keidl / Kemper / Seltzsam / Stocker (2002), S. 304ff.
[120] Vgl. Bellwood / Clément / Ehnebuske / Hately / Hondo / Husband / Januszewski / Lee / Munter / von Riegen (2002), S. 174ff.; Kreger (2001), S. 20.
[121] Vgl. Cerami (2002) , S. 158ff.; Chappell / Jewell / Dalheimer (2003), S. 110ff.; Bellwood / Clément / Ehnebuske / Hately / Hondo / Husband / Januszewski / Lee / Munter / von Riegen (2002).

anhand geographischer Parameter als auch anhand der Unternehmensbranche möglich.[122]

- Grüne Seiten: Die Informationen umfassen den technischen Datenbestand, der innerhalb des Verzeichnisses gespeichert wird. Daher werden hier Spezifikationsdokumente oder Referenzen auf diese als auch die Verbindungsendpunkte der einzelnen Web Services abgelegt, über die der Dienstnutzer den Dienst aktivieren kann. Die Spezifikationsdokumente selbst sind in der Regel in WSDL verfasst.

Diese aufgezeigten Daten werden in einer speziellen Datenstruktur innerhalb des Verzeichnisses abgelegt. Diese Datenstruktur kann man analog zur Struktur des WSDL-Dokuments in die beiden Teile der Implementierungs- und der Schnittstellenbeschreibung splitten. Die Abbildung erfolgt in Form eines XML-Dokuments und enthält vier zentrale Datenelemente.

In der Implementierungsbeschreibung (`businessEntity`) werden die Daten der weißen Seiten niedergelegt. Darüber hinaus finden sich hier Daten der gelben Seite wieder, die sich aus einer Kategorisierung des Dienstanbieters ergeben. Eine Zuordnung der angebotenen Dienste und somit die Speicherung der restlichen Daten der gelben Seite erfolgt in der informellen Dienstbeschreibung (`businessService`), die in der Implementierungsbeschreibung als Kindelement enthalten ist. Die technischen Details der konkreten Implementierung und somit ein Teil der Daten der grünen Seiten werden als Implementierungsorte (`bindingTemplates`) gespeichert. Wesentliches Element der Speicherung ist der Verbindungsendpunkt, also die Adresse des Web Service, über den der Dienst aktiviert werden kann. Dabei sind die Implementierungsorte der informellen Dienstbeschreibung als Kindelemente zugeordnet. Darüber hinaus enthält der Implementierungsort eine Referenz auf die Schnittstellenbeschreibung (`tModel`), in der die Dienstschnittstelle in Form eines WSDL-Dokuments oder aber als Referenz auf ein solches Dokument hinterlegt wird. Aufgrund der Hinterlegung als Referenz kann die Schnittstellenbeschreibung innerhalb verschiedener Implementierungsbeschreibungen wieder verwendet werden. Dadurch wird die Implementierung eines Dienstes auf Basis der gleichen Schnittstellensignatur durch verschiedene Dienstanbieter ermöglicht.[123]

Abbildung 22 ordnet die Klassen von Informationen der UDDI zugrunde liegenden Datenstruktur zu.

---

[122] Ein konkretes Beispiel für eine solche vorgesehene Taxonomie ist die Kategorisierung nach Ländercodes nach ISO 3166; vgl. Wojciechoski / Weinhardt (2002), S. 108.

[123] Vgl. Bellwood / Clément / Ehnebuske / Hately / Hondo / Husband / Januszewski / Lee / Munter / von Riegen (2002), S. 30ff.; Badach / Rieger / Schmauch (2003), S. 340.

Struktur \ Klasse	Weiße Seiten	Gelbe Seiten	Grüne Seiten
businessEntity	+	+	-
businessService	-	+	-
bindingTemplate	-	-	+
tModel	-	-	+

Abbildung 22: Zuordnung der Daten auf die Datenstruktur bei UDDI

Abbildung 23 veranschaulicht die Schachtelung der Datenelemente innerhalb der UDDI Datenstruktur. Auch wird die aufgezeigte Referenzierung auf die Schnittstellenbeschreibung aus den Implementierungsorten heraus deutlich. Eine komplette Darstellung der UDDI Datenstruktur in der UML-Notation ist bei Keidl et al. zu finden.[124]

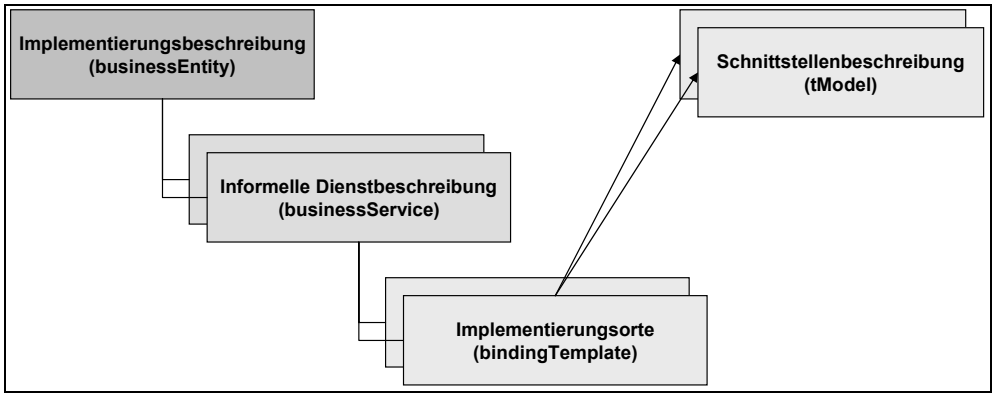

Abbildung 23: Beziehungen innerhalb der UDDI Datenstruktur

Zugriff

Der Zugriff auf die Daten erfolgt über standardisierte Schnittstellen, die die Grundfunktionalitäten des Einfügens, des Änderns, des Löschens und des Suchens abdecken. Die ersten drei Grundfunktionalitäten werden über Datenmodifikations-schnittstellen realisiert, deren Spezifikation als Publishing-API bezeichnet wird. Die Grundfunktionalität des Suchens wird durch die Inquiry-API realisiert. Neben diesen Zugriff über Application Programming Interfaces besteht optionalen bei manchen Betreibern des Verzeichnisdienstes die Möglichkeit, über nicht standardisierte Weboberflächen die Grundfunktionalitäten von UDDI in Anspruch zu nehmen. Für eine ausführliche Einführung in die Zugriffsmöglichkeiten sei an dieser Stelle auf die Spezifikation beziehungsweise auf Keidl verwiesen.[125]

[124] Vgl. Keidl / Kemper / Seltzsam / Stocker (2002), S. 308.
[125] Vgl. Bellwood / Clément / Ehnebuske / Hately / Hondo / Husband / Januszewski / Lee / Munter / von Riegen (2002), S. 62ff.; Keidl / Kemper / Seltzsam / Stocker (2002), S. 313ff.

3.6 Werkzeugunterstützung

Nachdem nun die relevanten Spezifikationen für die einzelnen Anforderungen im Protokollstapel von Web Services aufgezeigt wurden, wird im nachfolgenden Abschnitt die Werkzeugunterstützung der einzelnen Ebenen näher betrachtet. Dabei werden zuerst ein Überblick der Arten der Werkzeugunterstützung gegeben und danach auf jeder Ebene die Werkzeugunterstützung untersucht.

3.6.1 Überblick

Im Umfeld der Web Service Innovation sind eine Vielzahl von kostenlosen und kostenpflichtigen Werkzeugen für nahezu jede Plattform und Application Programming Interfaces (APIs) für jede gängige Programmiersprache verfügbar. Dabei setzen die Werkzeuge immer auf den APIs auf, die die grundlegenden Mechanismen zur Verfügung stellen, und ergänzen eine textuelle oder grafische Benutzeroberfläche. Teilweise wird der Funktionsumfang, der durch die grundlegenden APIs zur Verfügung gestellt wird, noch erweitert.

Nachfolgend werden die APIs und Werkzeuge für die Basistechnologien bei Web Services ausschnittsweise aufgeführt. Da alle Basistechnologien im Umfeld von Web Services auf XML aufsetzen, wird ein besonderer Fokus auf die verfügbaren APIs und die Werkzeugunterstützung von XML gelegt.

3.6.2 Inhaltsbeschreibung - XML

Basisbaustein für die Verarbeitung und den Zugriff auf XML-Dokumente sind die XML-Prozessoren, die auch als XML-Parser bezeichnet werden. Dabei handelt es sich um Softwaremodule, die XML-Dokumente einlesen und einen Zugang auf die Strukturen und Inhalte von XML-Dokumenten gestatten. Die Zugriffsmöglichkeiten für die Verarbeitung und die Analyse von XML-Dokumenten werden dabei über standardisierte Application Programming Interfaces (APIs) der nutzenden Anwendung zur Verfügung gestellt.[126] Es lassen sich prinzipiell mit dem Document Object Model (DOM) und dem Simple API for XML (SAX) zwei Arten von APIs unterscheiden, die im Folgenden kurz skizziert werden sollen.

[126] Vgl. Goldfarb / Prescod (2002), S. 50f.

Konkrete Umsetzungen der vorgestellten APIs und somit XML-Parser sind für alle wichtigen Programmiersprachen verfügbar.[127]

XML-Parser, die die API von DOM realisieren, repräsentieren XML-Dokumente durch Baumstrukturen, die die hierarchischen Strukturen innerhalb der Dokumente abbilden. Dazu werden in dieser Baumstruktur Element-, Attribut- und Inhaltsknoten unterschieden, denen jeweils Zugriffsmethoden u. a. zum Lesen, Suchen, Löschen und Schreiben der Knoteninhalte zugeordnet wurden.[128] Jedoch wird vor dem Zugriff das komplette XML-Dokument analysiert und in eine statische Repräsentation in Form eines Baumes überführt. Dies führt gerade bei umfangreichen XML-Dokumenten zu einem hohen Speicherbedarf und einer gewissen Verzögerung, bis das XML-Dokument für den Zugriff bereitsteht.[129]

Um diese Nachteile zu umgehen, hat sich mit SAX eine zweite Variante für den Zugriff in der Praxis etabliert. SAX zählt zu den ereignisbasierten APIs, die bei jedem Auftreten eines XML-Konstrukts (Start-Marke, End-Marke, Elementinhalt, etc.) während des Analysierens des XML-Dokuments ein Ereignis auslösen, welches die nutzende Anwendung abfangen und weiterverarbeiten muss.[130] Jedoch hat die auf ereignisbasierten Konzepten resultierende Effizienz von SAX negative Auswirkungen auf die Benutzerfreundlichkeit der zugrunde liegenden API.

Da XML-Prozessoren lediglich die grundlegenden Zugriffsmechanismen zur Verfügung stellen, haben sich zur Erweiterung dieser Basisfunktionalitäten Anfrage- und Transformationssprachen entwickelt. Diese bauen auf den XML-Prozessoren auf und unterstützen sowohl die Gewinnung von Informationen aus Datenbeständen als auch die Konvertierung von Datenbeständen aus einer Struktur in eine andere Struktur. Auch diese erweiterten Funktionalitäten werden über Application Programming Interfaces zur Verfügung gestellt.

Die durch das W3C entwickelte Extensible Stylesheet Language (XSL), die sowohl zur Formatierung als auch zur Transformation von XML-Dokumenten eingesetzt werden kann, bildet diese Erweiterungen in einer Spezifikation ab.[131] Bei der Spezifikation von XSL wurden die grundlegenden Konzepte der seit 1996 der etablierten Document Style Semantics and Specification Language (DSSSL)[132] übernommen, da diese bereits erfolgreich im Umfeld von

---

[127] Vgl. McLaughlin / Kersken / Key (2002), S. 111ff.
[128] Vgl. Wood (1998).
[129] Vgl. Laurent (2001), S. 199.
[130] Vgl. Morgenthal / Forge (2001), S. 56ff.
[131] Vgl. Harold (2001), S. 433ff.
[132] Die DSSSL Spezifikation besteht aus einer Transformations- und einer Stilsprache. Die Transformationssprache wandelt SGML-Dokumente, die Instanzen in einer bestimmten Strukturdefinition sind, in Instanzdokumente einer beliebigen anderen Strukturdefinition um. Die Stilsprache legt das Layout einzelner Bestandteile des SGML-Dokuments fest, wobei Ausgabeformate in Form von

SGML im Einsatz war. Die Umsetzung jedoch erfolgte in einer syntaktisch anderen Form.[133] Die Spezifikation von XSL besteht mit XML Path Language (XPath), XSL Transformations (XSLT) und XSL Formatting Objects (XSL-FO) aus drei aufeinander aufbauenden Spezifikationen.[134]

XPath wurde im November 1999 als Standard für eine Anfragesprache verabschiedet und dient der Adressierung von Teilen eines XML-Dokuments. So können nicht nur XML-Dokumente als Ganzes über Links angesprochen werden, sondern auch einzelne Abschnitte eines XML-Dokuments oder sogar konkrete Elemente.[135] Somit werden durch XPath Wegbeschreibungen in Form von Pfadangaben innerhalb von XML-Dokumente ermöglicht, die sich syntaktisch an die Notation von URIs halten und somit eine Navigation innerhalb der XML-Dokumente ermöglichen.[136] Jedoch erwies sich XPath in der Version 1.0 aufgrund der eingeschränkten Funktionalität und der schwachen Ausdruckkraft in Bezug auf die Darstellung von Abfragen als unzureichend. Daher wurde mit der Entwicklung von XQuery als neue Anfragesprache begonnen, die bis heute (Stand: Mai 2003) noch nicht abgeschlossen ist. In aktuellste Spezifikation liegt XQuery seit Mai 2003 vor.[137] XQuery vereinigt die herausragenden Eigenschaften existierender Anfragesprachen (allen voran XPath und XQL) und verbindet diese mit den bewährten Design von Anfragesprachen wie der Structured Query Language (SQL) und der Object Query Language (OQL). Parallel zur Spezifikation von XQuery wurde auch XPath weiterentwickelt und hat in der Version 2.0 deutlich an Mächtigkeit zugelegt. XPath hält nun neben diverser Datentypen auch eine Vielzahl von vordefinierten Funktionen bereit, die auf diesen Datentypen operieren. Somit verwundert es nicht, dass XPath zentraler Bestandteil der Spezifikation von XQuery ist und in der Spezifikation von XQuery über 80% einnimmt. Somit ist jede in XQuery spezifizierte Anfrage auch eine gültige Anfrage in XPath.[138]

XSLT wurde ebenfalls im November 1999 als Standard vom W3C verabschiedet und verwendet XPath als Anfragesprache. Die Spezifikation umfasst Vorschriften zur Transformation eines XML-Quelldokuments in ein beliebiges Zieldokument, wobei es sich beim Quelldokument um ein wohlgeformtes XML-Dokument handeln muss.[139] Dabei werden die konkreten Umwandlungsvorschriften in einem so genannten Stylesheet hinterlegt. Die

---

Portable Document Form (PDF), Postscript und Rich Text Format (RTF) unterstützt werden. Vgl. u. a. Behme / Mintert (2001), S. 123ff.

[133] Vgl. Behme / Mintert (2001), S. 134f.
[134] Vgl. Thompson (2002).
[135] Vgl. Goldfarb / Prescod (2002), S. 1005ff.
[136] Vgl. Clark / DeRose (1999).
[137] Vgl. Boag / Chamberlin / Fernandez / Florescu / Robie / Siméon (2003).
[138] Vgl. Berglund / Boag / Chamberlin / Fernandez / Kay / Robie / Siméon (2003); Malhotra / Melton / Robie / Walsh (2003).
[139] Vgl. Weitzel / Harder / Buxmann (2001), S 40f.

Durchführung einer Transformation wird dann von einem XSLT-Prozessor übernommen, der das Quelldokument und das Stylesheet einliest und dann das Stylesheet auf das Quelldokument anwendet. XSLT-Prozessoren unterscheiden sich also grundlegend von XML-Prozessoren, da XSLT-Prozessoren XML-Dokumente transformieren, XML-Prozessoren jedoch lediglich die Struktur und den Inhalt von XML-Dokumenten für andere Anwendungen zur Verfügung stellen.

XSL-FO wurde im Oktober 2001 als Standard vom W3C verabschiedet und beinhaltet Mechanismen zur Formatierung von XML-Dokumenten. Dazu werden XML-Quelldokumente mit so genannten Formatting Objects (FOs) ausgestattet, in dem ein XSL-FO-Stylesheet auf die Quelldokumente angewendet wird.[140] XSL-FO teilt die XML-Dokumente in verschiedene Bereiche auf. Dabei ist eine Überlappungen oder eine Schachtelung von verschiedenen Abschnitten möglich. Die einzelnen Teile werden dann mittels Formatting Objects formal beschrieben, indem beispielsweise Ränder und Abstände zwischen Inhalten festgelegt oder bestimmten Bereichen Hintergrundfarben zugewiesen werden können. Für weitere Ausführungen sei jedoch aufgrund der Komplexität auf die Spezifikation verwiesen.[141]

Die Werkzeuge setzen nun auf diesen Basisfunktionalitäten und Erweiterungen auf. Dabei lassen sich Werkzeuge zur Bearbeitung von XML-Dokumenten sowie für die Bearbeitung von Strukturdefinitionen unterscheiden.[142]

Im Bereich der Bearbeitung von XML-Dokumenten sind speziell XML-Editoren anzuführen, die die Einhaltung auf XML-spezifische Eigenschaften übernehmen. Dazu zählen beispielsweise die Prüfung auf Wohlgeformtheit und Gültigkeit, die Unterstützung bei der Elementauswahl sowie verschiedene Sichten (Quelltext, Strukturbaum) auf die zu bearbeiteten XML-Dokumente. Bei den Nutzern von XML-Editoren bilden die Gruppe der Anwender den Schwerpunkt, die die inhaltliche Arbeit an den XML-Dokumenten absolvieren. Daher wurde bei diesem Werkzeugen ein Hauptaugenmerk auf eine einfache Benutzerführung und die entsprechende Unterstützung durch Hilfemenüs gelegt.

Die Erstellung und Bearbeitung von Strukturdefinitionen werden durch so genannte Struktureditoren ermöglicht. Dabei werden durch die Struktureditoren sowohl Document Type Definitions (DTD) und XML Schemata als mögliche Spezifikationsgrundlagen unterstützt. Als Anwender werden hier schwerpunktmäßig Nutzer angesprochen, die die strukturellen Vorgaben für die darauf basierenden XML-Dokumente erstellen.

---

[140] Vgl. Bach (2000), S. 206ff.
[141] Vgl. Adler / Berglund / Caruso / Deach / Graham / Grosso / Gutentag / Milowski / Parnell / Richman / Zilles (2001).
[142] Vgl. Rawolle (2002), S. 64ff.

3.6.3 Dienstkommunikation - SOAP

SOAP selbst ist ein XML-basiertes und dadurch einfaches, sprach-, plattformunabhängiges Kommunikationsprotokoll zum Austausch strukturierter Informationen. Da zu übertragene Nachricht wird in Form eines XML-Dokuments übermittelt. Daher setzen die Application Programming Interfaces auf den Basisbaustein zur Verarbeitung und zum Zugriff, den XML-Prozessoren oder XML-Parsern, auf.

Dabei stellen die APIs Funktionalitäten zur Verfügung, die auf der einen Seite die Erzeugung und die Verarbeitung von Nachrichten ermöglichen. Auf der anderen Seite werden Funktionalitäten angeboten, die die Übertragung und somit das Senden und Empfangen von Nachrichten übernehmen. Diese Application Programming Interfaces sind für alle gängigen Programmiersprachen erhältlich.

Werkzeuge setzen auf diesen Basisfunktionalitäten auf, und bieten folgende Funktionalitäten an:

- Generierung von Proxies aus Schnittstellenbeschreibungen, die die Verarbeitung und die Übermittlung von Nachrichten übernehmen
- Generierung von Testumgebungen, die bereits umgesetzte Schnittstellen auf Funktionalität und Fehlerfreiheit überprüfen
- Kontrolle und Überwachung des Nachrichtenverkehrs zwischen Sender und Empfänger, so genannte Monitoring-Tools

Zu nennen sind hier beispielsweise Apache SOAP[143], Apaches Extensible Interaction System (AXIS)[144], das Web Services Toolkit (WSTK)[145] von IBM für die Programmiersprache Java oder aber das SOAP Toolkit[146] von Microsoft für die Programmiersprachen C++ und C#. Auch für die Skriptsprache PHP ist mit PHPSOAP[147] eine Umsetzung verfügbar. Eine aktuelle Zusammenstellung auf der Webseite von SoapWare.org verzeichnet zur Zeit über 90 verschiedene Umsetzungen von Application Programming Interfaces und Werkzeugen für die Dienstkommunikation auf Basis von SOAP.[148]

[143] Vgl. http://ws.apache.org/soap/
[144] Vgl. http://ws.apache.org/axis/
[145] Vgl. http://www.alphaworks.ibm.com/tech/webservicestoolkit
[146] Vgl. http://msdn.microsoft.com/soap/
[147] Vgl. http://phpsoaptoolkit.sourceforge.net/phpsoap/
[148] Vgl. Kulchenko (2003).

3.6.4 Dienstbeschreibung - WSDL

WSDL spezifiziert die Struktur eines XML-Dokuments, das die Beschreibung eines Web Services und somit die Funktion einer Interface Definition Language (IDL) übernimmt. Somit setzen auch die Application Programming Interfaces auf dem XML-Prozessoren als Basisbaustein für die Erstellung und Verarbeitung von XML-Dokumenten auf.

Die Application Programming Interfaces stellen dabei analog zu SOAP Funktionalitäten zur Verfügung, die die Erstellung und die Bearbeitung von WSDL-Dokumenten ermöglichen.

Werkzeuge setzen wiederum auf diesen Basisfunktionalitäten auf und lassen sich in folgende Aufgabenklassen gliedern:

- Graphische Editoren zur Erstellung und Bearbeitung von WSDL-Dokumenten: Diese Editoren übernehmen auf der einen Seite die gleichen Aufgaben wie XML-Editoren. Auf der anderen Seite überprüfen Sie die Struktur des WSDL und bieten eine spezielle Elementauswahl an, die nur bei WSDL-Dokumenten gültige Elemente anbietet.
- Tools zur Generierung von WSDL-Dokumenten aus existierenden Quellcodes, beispielsweise aus Java-Klassen oder Enterprise Java Beans: Dieses Vorgehen kann dabei als Botton-Up-Vorgehen bezeichnet werden, da ausgehend von einer konkreten Umsetzung die Schnittstellenbeschreibung erzeugt wird. Der Anwender bei der Umsetzung seine vertraute Entwicklungsumgebung und die Programmiersprache beibehalten und muss sich nicht mit der Struktur eines WSDL-Dokuments auskennen.
- Tools zur Generierung von Proxy-Klassen (Service Proxies auf der Clientseite und Service Implementation Templates auf der Serverseite) als Rahmenklassen, die die Erstellung und die Übertragung von Nachrichten übernehmen: In diesem Fall kann man von einem Top-Down-Vorgehen sprechen, da ausgehend von einer Schnittstellenbeschreibung die entsprechenden Umsetzung der Rahmenklassen in jeder beliebigen Programmiersprache ermöglicht wird. In Kombination mit dem Botton-Up-Vorgehen muss der Anwender weder Kenntnisse in Bezug auf WSDL noch Kenntnisse in Bezug auf den Aufbau der Nachricht und der Übertragung aneignen.
- Tools zur Erstellung von Testumgebungen, die dann zum Überprüfen bereits umgesetzter Funktionalitäten genutzt werden können.

Auch hier sind analog zu der Werkzeugunterstützung von SOAP die Produkte von beispielsweise Apache SOAP[149], Apaches Extensible Interaction System (AXIS)[150], das Web Services Toolkit (WSTK)[151] von IBM für die Programmiersprache Java zu nennen. Auch die .NET-Plattform[152] von Microsoft bietet die aufgeführten Funktionalitäten für die Programmiersprachen C++ und C# sowie Werkzeuge an. Beispielhaft sollen die aufgeführten Funktionalitäten mit konkreten Produkten belegt werden.

WSDL4J aus dem Produkt WSTK ermöglicht die Erstellung und Verarbeitung von WSDL-Dokumenten auf Basis der Programmiersprache Java, in dem es eine Application Programming Interface für diese Aufgabe realisiert. WSDL2Java aus dem Produkt AXIS ermöglicht die Generierung von Proxy-Klassen in der Programmiersprache Java aus einem WSDL-Dokument. Auch gestattet das durch AXIS ausgelieferte Servlet die Erzeugung eines WSDL-Dokuments aus einer Java-Klasse zur Laufzeit der Anwendung.

3.6.5 Dienstverzeichnis - UDDI

Bei UDDI werden der Zugriff auf die im Verzeichnis gespeicherten Daten durch Application Programming Interfaces und Werkzeuge unterstützt.

Dabei setzen die Application Programming Interfaces die Funktionalitäten gemäß der Spezifikation für den Zugriff um. Die Publishing-API übernimmt das Einfügen, das Ändern und das Löschen von Datensätzen und die Inquiry-API die Suche innerhalb des Verzeichnisses.[153]

Werkzeuge setzen auf diesen Grundfunktionalitäten auf und automatisieren den Zugriff auf das Verzeichnis, indem der Nutzer durch textuelle oder graphische Oberflächen unterstützt wird. Dazu zählen beispielsweise Assistenten, die den Nutzer eine Abfolge von Formularen anbieten, die die Eingabe der nötigen Informationen zur Veröffentlichung bzw. zur Modifikation ermöglichen. Die Änderung der Daten im jeweiligen Verzeichnis verläuft dann vollautomatisch und ohne jegliche Benutzerinteraktion. Auch sind Assistenten verfügbar, auf deren Basis eine komfortable Schnittstelle zur Suche zur Verfügung stellen. Diese ermöglichen die Eingabe von Suchparametern und bereiten dann die Suchergebnisse in graphischer Form dem Nutzer auf.

---

[149] Vgl. http://ws.apache.org/soap/
[150] Vgl. http://ws.apache.org/axis/
[151] Vgl. http://www.alphaworks.ibm.com/tech/webservicestoolkit
[152] Vgl. http://www.mircosoft.com/net/
[153] Vgl. Bellwood / Clément / Ehnebuske / Hately / Hondo / Husband / Januszewski / Lee / Munter / von Riegen (2002), S. 62ff.; Keidl / Kemper / Seltzsam / Stocker (2002), S. 313ff.

Solche Assistenten sind beispielsweise in der .NET-Plattform von Microsoft vertreten und kapseln die Kommunikation zwischen den Nutzer und dem Verzeichnis, die auf Basis der aufgezeigten API-Spezifikation der Verzeichnisse erfolgt. Der Nutzer selbst erhält eine graphische Oberfläche, die die Eingabe der für die Anforderung notwendigen Daten ermöglicht. Auch bietet der WebSphere Studio Application Developer mit dem UDDI-Explorer ein solches Werkzeug an, welches die Veröffentlichung in einem UDDI Verzeichnis unterstützt.[154]

[154] Vgl. Badach / Rieger / Schmauch (2003), S. 342f.

4 Fazit

Dieser Arbeitsbericht stellt eine Einführung in die Thematik Web Services und deren Umfeld dar. Im zweiten Kapitel wurde ausgehend von einer Begriffsdefinition für Web Services die Service Oriented Architecture (SOA) als grundlegendes Architekturmodell und die möglichen Kommunikationsparadigmen innerhalb des Architekturmodells vorgestellt. Daran anschließend wurde eine Abgrenzung zu Konzepten zur Realisierung von verteilten Anwendungen auf der einen Seite und zu Middleware auf der anderen Seite und somit zu verwandten Konzepten vorgenommen. Da es sich bei Web Services um eine Schnittstellentechnologie handelt, schließt der Grundlagenabschnitt mit einer Diskussion von Schlüsselfaktoren, die eine Bewertung hinsichtlich der Eignung von Web Services als Schnittstellentechnologie für eine angebotene Dienstleistung ermöglichen.

Im dritten Kapitel wurde dann der Protokollstapel eingeführt, anhand dessen eine Systematisierung und Anordnung der Standards im Web Services Umfeld möglich sind, und der die Basisfunktionalitäten der grundlegenden Architektur realisiert. Die im Web Services Umfeld diskutierten Standards wurden eingeordnet und die drei Kernstandards SOAP, WSDL und UDDI erläutert. Für diese Einführung in die Kernstandards war eine Einführung in die Metasprache XML unerlässlich, da alle Kernstandards auf dieser Metasprache basieren und diese Nutzen. Das dritte Kapitel schließt mit einer Untersuchung der Werkzeugunterstützung der Kernstandards.

Zusammenfassend lässt sich festhalten, dass das grundlegende Konzept von Web Services als unkompliziert eingestuft werden kann. Auch sind für alle Kernstandards für alle Plattformen und Programmiersprachen Application Programming Interfaces verfügbar. Dies führt dazu, dass Web Services im Vergleich zu den anderen Konzepten zur Realisierung von verteilten Anwendungen relativ niedrige Einstiegsbarrieren aufweisen. Auch zeichnen sich Web Services durch eine hohe Interoperabilität aus, da etablierte Internetprotokolle und XML als Basis für die Kernstandards genutzt werden. Web Services sind daher prinzipiell von jeden digitalen Endgerät aus nutzbar und weisen daher einen gewissen Grad von Ubiquität auf. Als wesentlich für die Durchsetzung ist auch einzustufen, dass global führende Softwarefirmen (SAP, IBM; Sun, Microsoft, etc.) die Entwicklung von Spezifikationen im Web Services Umfeld vorantreiben und auch die aufgezeigten Kernstandards im Kern auf Vorschlägen dieser Softwarefirmen fußen.

Jedoch wurden auch Problembereiche im Rahmen dieses Arbeitsberichts angesprochen. So fehlen zur Zeit noch Lösungsansätze für die Problemfelder der Abrechnung, der

Transaktionssicherheit, der Sicherheit der Datenübertragung. Auch für die Kombination von Web Services zu einem komplexen Geschäftsprozess sind bisher nur grundlegende Spezifikationen zur Beschreibung solcher Workflows vorhanden. Daher werden sich die nächsten Forschungsaufgaben mit der Lösung gerader dieser Problempunkte beschäftigen.

Auch sind anhand der aufgezeigten Schlüsselfaktoren für den Einsatz von Web Services als weitere Forschungsleistung Anwendungsgebiete zu identifizieren, die als besonders geeignet für den Einsatz von Web Services als Schnittstellentechnologie angesehen werden können. In diesen Anwendungsgebieten ist dann der Einsatz von Web Services sowie entsprechende Vor- und Nachteile zu diskutieren.

Literaturverzeichnis

Adler / Berglund / Caruso / Deach / Graham / Grosso / Gutentag / Milowski / Parnell / Richman / Zilles (2001): Adler, S. / Berglund, A. / Caruso, J. / Deach, S. / Graham, T. / Grosso, P. / Gutentag, E. / Milowski, A. / Parnell, S. / Richman, J. / Zilles, S., Extensible Stylesheet Language (XSL) Version 1.0, http://www.w3.org/TR/xsl/, Abruf am 30.04.2003.

Austin / Barbir / Ferris / Garg (2002): Austin, D. / Barbir, A. / Ferris, C. / Garg, S., Web Services Architecture Requirements, http://www.w3.org/TR/2002/WD-wsa-reqs-20021011, Abruf am 21.05.2003.

Bach (2000): Bach, M.: XSL und XPath - verständlich und praxisnah: Transformationen und Ausgabe von XML-Dokumentationen mit XSL, Boston 2000.

Badach / Rieger / Schmauch (2003): Badach, A. / Rieger, S. / Schmauch, M.: Web-Technologien: Architekturen, Konzepte, Trends, München [u.a.] 2003.

Balen / Elenko (2000): Balen, H. / Elenko, M.: Distributed object architectures with CORBA, Cambridge [u.a.] 2000.

Basha (2002): Basha, S. J.: Professional Java web services, Birmingham 2002.

Behme / Mintert (2001): Behme, H. / Mintert, S.: XML in der Praxis: professionelles Web-Publishing mit der Extensible Markup Language, 2, München [u.a.] 2001.

Beimborn / Mintert / Weitzel (2002): Beimborn, D. / Mintert, S. / Weitzel, T.: WI - Schlagwort - Web Services und ebXML. In: Wirtschaftsinformatik 44 (2002) 3, S. 277-280.

Bellwood / Clément / Ehnebuske / Hately / Hondo / Husband / Januszewski / Lee / Munter / von Riegen (2002): Bellwood, T. / Clément, L. / Ehnebuske, D. / Hately, A. / Hondo, M. / Husband, Y. L. / Januszewski, K. / Lee, S. B. / Munter, J. / von Riegen, C., UDDI Version 3.0 - UDDI Spec Technical Committee Specifaction, http://uddi.org/pubs/uddi-v3.00-published-20020719.pdf, Abruf am 01.09.2003.

Berglund / Boag / Chamberlin / Fernandez / Kay / Robie / Siméon (2003): Berglund, A. / Boag, S. / Chamberlin, D. / Fernandez, M. F. / Kay, M. / Robie, J. / Siméon, J., XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath20/, Abruf am 19.06.2003.

Bettag (2001): Bettag, U.: Web Services. In: Informatik Spektrum 5 (2001) 24, S. 302-304.

Birrell / Nelson (1984): Birrell, A. D. / Nelson, B. J.: Implementing remote procedure calls. In: ACM Transaction on Computer Systems 2 (1984) 1, S. 39-59.

Boag / Chamberlin / Fernandez / Florescu / Robie / Siméon (2003): Boag, S. / Chamberlin, D. / Fernandez, M. F. / Florescu, D. / Robie, J. / Siméon, J., XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/, Abruf am 17.06.2003.

Box / Ehnebuske / Kakivaya / Layman / Mendelsohn (2002): Box, D. / Ehnebuske, D. / Kakivaya, G. / Layman, A. / Mendelsohn, N., Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/SOAP/, Abruf am 18.12.2002.

Box / Skonnard / Lam (2000): Box, D. / Skonnard, A. / Lam, J. F.: Essential XML: beyond markup, Boston 2000.

Bray / Hollander / Layman (1999): Bray, T. / Hollander, D. / Layman, A., Namesspaces in XML, http://www.w3.org/TR/REC-xml-names/, Abruf am 30.04.03.

Bray / Paoli / Sperberg-McQueen / Maler (2000): Bray, T. / Paoli, J. / Sperberg-McQueen, C. M. / Maler, E., Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/REC-xml, Abruf am 30.04.2003.

Britton (c2001): Britton, C.: IT architectures and middleware: strategies for building large, integrated systems, Boston, Mass. [u.a.] c2001.

Buchmann / Casati / Fiege / Hsu / Shan (2002): Buchmann, A. / Casati, F. / Fiege, L. / Hsu, M. / Shan, M.: Proceedings of the Third VLDB Workshop on Technologies for E-Services, Lecture Notes in Computer Science 2444, Hong Kong 2002.

Burghardt / Gehrke / Schumann (2003a): Burghardt, M. / Gehrke, N. / Schumann, M.: Implikationen kommerzieller Web Services, In: Eckstein, R. / Tolksdorf, R.: XMIDX 2003 - XML-Technologien für Middleware - Middleware für XML-Anwendungen, Köln 2003, S. 71-82.

Burghardt / Gehrke / Schumann (2003b): Burghardt, M. / Gehrke, N. / Schumann, M.: Eine Architektur zur Abrechnung von Web Services, In: Eckstein, R. / Tolksdorf, R.: XMIDX 2003 - XML-Technologien für Middleware - Middleware für XML-Anwendungen, Köln 2003, S. 45-56.

Cerami (2002): Cerami, E.: Web services essentials, Beijing [u.a.] 2002.

Chappell / Jewell / Dalheimer (2003): Chappell, D. A. / Jewell, T. / Dalheimer, M. K.: Java Web Services, Beijing [u.a.] 2003.

Chaudhri (2003): Chaudhri, A. B.: Web, web-services, and data-base systems: revised papers, Berlin [u.a.] 2003.

Chinnici / Gudgin / Moreau / Weerawarana (2003): Chinnici, R. / Gudgin, M. / Moreau, J. / Weerawarana, S., Web Services Description Language (WSDL) Version 1.2, http://www.w3.org/TR/wsdl12/, Abruf am 01.09.2003.

Christensen / Curbera / Meredith / Weerawarana (2002): Christensen, E. / Curbera, F. / Meredith, G. / Weerawarana, S., Web Services Description Language (WSDL) 1.1..., http://www.w3.org/TR/wsdl, Abruf am 25.06.2002.

Clark / DeRose (1999): Clark, J. / DeRose, S., XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath, Abruf am 30.04.2003.

Coulouris / Dollimore / Kindberg (2002): Coulouris, G. / Dollimore, J. / Kindberg, T.: Verteilte Systeme: Konzepte und Design, 3, München 2002.

Curbera / Duftler / Khalaf / Nagy / Mukhi / Weerawarana (2002): Curbera, F. / Duftler, M. / Khalaf, R. / Nagy, W. / Mukhi, N. / Weerawarana, S.: Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. In: IEEE Internet computing 6 (2002) 2, S. 86-93.

Eddy / Delong (2001): Eddy, S. E. / Delong, B.: XML in plain English, Foster City, Calif. [u.a.] 2001.

Fessenbecker (2002): Fessenbecker, M.: Web Services - Revolution für die B2B-Integration?. In: IM 17 (2002) 3, S. 47-50.

Galbraith / Hankison / Hiotis / Prasad / Trivedi / Whitney D. (2002): Galbraith, B. / Hankison, W. / Hiotis, A. J. M. / Prasad, D. / Trivedi, R. / Whitney D.,: Professional Web Services Security, 2002.

Genussa (1999): Genussa, P.: Evolution and Use of Generic Markup Languages, In: Möhr, W.: SGML und XML: Anwendungen und Perspektiven, Berlin [u.a.] 1999, S. 27-50.

Goldfarb / Prescod (2002): Goldfarb, C. F. / Prescod, P.: XML handbook, Upper Saddle River, NJ 2002.

Gottschalk / Graham / Kreger / Snell (2002): Gottschalk, K. / Graham, S. / Kreger, H. / Snell, J.: Introduction to Web services architecture. In: IBM systems journal 41 (2002) 2, S. 170-177, insges. 8 S..

Graham / Simeonov / Boubez (2002): Graham, S. / Simeonov, S. / Boubez, T.: Building web services with Java: making sense of XML, SOAP, WSDL, and UDDI, Indianapolis, Ind. 2002.

Gudgin / Hadley / Mendelsohn / Moreau / Nielsen (2003a): Gudgin, M. / Hadley, M. / Mendelsohn, N. / Moreau, J. / Nielsen, H. F., SOAP Version 1.2 Part 1: Messaging Framework, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/, Abruf am 01.09.2003. (a)

Gudgin / Hadley / Mendelsohn / Moreau / Nielsen (2003b): Gudgin, M. / Hadley, M. / Mendelsohn, N. / Moreau, J. / Nielsen, H. F., SOAP Version 1.2 Part 2: Adjuncts, http://www.w3.org/TR/2003/REC-soap12-part2-20030624/, Abruf am 01.09.2003. (b)

Haase (2001): Haase, O.: Kommunikation in verteilten Anwendungen, München, Wien 2001.

Hansch / Kuhlins / Schader (2002): Hansch, M. / Kuhlins, S. / Schader, M.: XML-Schema. In: Informatik Spektrum 25 (2002) 5, S. 363-366.

Hansen / Neumann (2002): Hansen, H. R. / Neumann, G.: Grundlagen betrieblicher Informationsverarbeitung, 8, Stuttgart 2002.

Harold (2001): Harold, E. R.: XML bible, New York, NY [u.a.] 2001.

Hars / Schlüter-Langdon (2002): Hars, A. / Schlüter-Langdon, C.: Chancen und Risiken für verteilte Informationssysteme. In: IM 17 (2002) 3, S. 13-19.

Hartmann / Flinn / Beznosov / Kawamoto (2003): Hartmann, B. / Flinn, D. J. / Beznosov, K. / Kawamoto, S.: Mastering Web Services Security, Indianapolis 2003.

Hoch (1996): Hoch, T.: Einsatz der Informationsverarbeitung bei Business Process Reengineering: Elemente eines Vorgehensmodells für Dienstleistungsunternehmen, Wiesbaden 1996.

Hoidn / Jungclaus (2002): Hoidn, H. P. / Jungclaus, R.: Web Services aus Sicht der Unternehmensarchitektur. In: Information Management & Consulting 17 (2002) 3, S. 31-35.

Holzner (2002): Holzner, S.: XSLT - Anwendung und Referenz: XML-Transformationen, XPath, Einsatz mit Java, JSP und ASP, München 2002.

Itter (1999): Itter, R.: Internet-basierte Informationssysteme in betrieblichen Prozessen: Einsatzbereiche und Vorgehensmodell, Lohmar [u.a.] 1999.

Jeckle (2002): Jeckle, M., Web Services, http://www.jeckle.de/webServices/index.html, Abruf am 26.05.2003.

Keidl / Kemper / Seltzsam / Stocker (2002): Keidl, M. / Kemper, A. / Seltzsam, S. / Stocker, K.: Web Services, In: Rahm, E. / Vossen, G.: Web & Datenbanken - Konzepte, Architekturen, Anwendungen, Heidelberg 2002, S. 293-335.

Klyne / Carroll (2003): Klyne, G. / Carroll, J. J., Resource Description Framework (RDF), http://www.w3.org/TR/rdf-concepts/, Abruf am 01.09.2003.

Knuth (2002): Knuth, M.: Web Services: Einführung und Übersicht, Frankfurt 2002.

Kreger (2001): Kreger, M., Web Services Conceptual Architecture, http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf, Abruf am 20.05.2001.

Krüger / Deutschmann (2002): Krüger, G. / Deutschmann, J.: Lehr- und Übungsbuch Telematik: Netze - Dienste - Protokolle, 2, München [u.a.] 2002.

Kulchenko (2003): Kulchenko, P., SOAP Implementations, http://www.soapware.org/directory/4/implementations, Abruf am 01.09.2003.

Kurbel / Rautenstrauch / Rödding / Scheuch (1995): Kurbel, K. / Rautenstrauch, C. / Rödding, T. / Scheuch, R.: Funktionsintegration in heterogenen verteilten Informationssystem - innovative Konzepte und Fallstudien, In: König, W.: Wirtschaftsinformatik 1995, Heidelberg 1995, S. 445-460.

Langner (2002): Langner, T.: Verteilte Anwendungen mit Java: Enterprise-Architekturen im Web mit CORBA, XML/SOAP, JSP, (E)JB und JDBC, München 2002.

Langner (2003): Langner, T.: Web Services mit Java: Neuentwicklung und Refactoring in der Praxis, München 2003.

Laurent (2001): Laurent, S. S.: XML: a primer, New York, NY [u.a.] 2001.

Light (1997): Light, R.: Presenting XML, Indianapolis, Ind. 1997.

Löwer / Picot (2002): Löwer, U. M. / Picot, A.: Web Services - Technologie - Hype oder Strategie-Faktor. In: Information Management & Consulting 17 (2002) 3, S. 20-25.

Mahmoud (2000): Mahmoud, Q. H.: Distributed programming with Java: [sockets, RMI, CORBA, mobile agents], Greenwich, Conn. 2000.

Malhotra / Melton / Robie / Walsh (2003): Malhotra, A. / Melton, J. / Robie, J. / Walsh, N., XQuery 1.0 and XPath 2.0 Functions and Operators, http://www.w3.org/TR/2002/WD-xquery-operators-20020816/, Abruf am 19.06.2003.

Martin / Burstein / Lassila / Paolucci / Payne / McIlraith (2002): Martin, D. / Burstein, M. / Lassila, O. / Paolucci, M. / Payne, T. / McIlraith, S., Describing Web Services using DAML-S and WSDL, http://www.daml.org/services/daml-s/0.7/daml-s-wsdl.html, Abruf am 01.09.2003.

McLaughlin / Kersken / Key (2002): McLaughlin, B. / Kersken, S. / Key, J.: Java XML, Dt. Ausg. der 2. Aufl, Beijing [u.a.] 2002.

Mertens (2003): Mertens, P.: XML-Schema-Spezifikation, In: Mertens, P.: XML Komponenten, Berlin [u.a.] 2003, S. 257-291.

Michel (1999): Michel, T.: XML kompakt - Struktur, Layout, Verweise: eine praktische Einführung, Wien 1999.

Mitra (2003): Mitra, N., SOAP Version 1.2 Part 0: Primer, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, Abruf am 01.09.2003.

Mohan (2002): Mohan, C.: Dynamic E-Business: Trends in Web Services, In: Buchmann, A. / Casati, F. / Fiege, L. / Hsu, M. / Shan, M.: Proceedings of the third VLDB workshop on Technologies for E-Services, TES 2002, Hong Kong 2002, S. 1-5.

Monson-Haefel / Kalle Dalheimer (2001): Monson-Haefel, R. / Kalle Dalheimer, M.: Enterprise JavaBeans, 1. Aufl, Beijing [u.a.] 2001.

Morgenthal / Forge (2001): Morgenthal, J. P. / Forge, B. L.: Enterprise application integration with XML and Java, Upper Saddle River, NJ [u.a.] 2001.

Müller-Stewens / Eymann / Kreutzer (2003): Müller-Stewens, G. / Eymann, T. / Kreutzer, M.: Telematik- und Kommunikationssysteme in der vernetzten Wirtschaft, München [u.a.] 2003.

Nagy / Curbera / Weerawaranna (2000): Nagy, W. A. / Curbera, F. / Weerawaranna, S., The Advertisement and Discovery of Services (ADS) protocol for Web services, http://www-106.ibm.com/developerworks/web/library/ws-ads.html, Abruf am 01.09.2003.

Newcomer (2002): Newcomer, E.: Understanding web services: XML, WSDL, SOAP, and UDDI, Boston, [Mass. u.a.] 2002.

Orfali / Harkey / Edwards (1994): Orfali, R. / Harkey, D. / Edwards, J.: Essential client/server survival guide, New York [u.a.] 1994.

Orfali / Harkey / Edwards (1997): Orfali, R. / Harkey, D. / Edwards, J.: Abenteuer Client/Server: the essential client/server survival guide, Bonn [u.a.] 1997.

Phillips (2000): Phillips, L. A.: Using XML, Special ed, Indianapolis, Ind. 2000.

Priesnitz / Teille (2003): Priesnitz, M. / Teille, K.: XML als Grundlage für standardisierte Internetschnittstellen, In: Mertens, P.: XML Komponenten, Berlin [u.a.] 2003, S. 9-27.

Rawolle (2002): Rawolle, J.: Content management integrierter Medienprodukte: ein XML-basierter Ansatz, Wiesbaden 2002.

Rawolle / Burghardt (2002): Rawolle, J. / Burghardt, M.: Web Services - eine Alternative für die zwischenbetriebliche Integration. In: IS Report 6 (2002) 2, S. 40-46.

Rieck / Dostal / Schandl / Sieb (2003): Rieck, M. / Dostal, W. / Schandl, R. / Sieb, C.: Web Services - vom Hype zum realen Einsatz im Finanzsektor, In: Mertens, P.: XML-Komponenten in der Praxis, Berlin 2003, S. 293-328..

Saint Laurent / Johnston / Dumbill (2001): Saint Laurent, S. / Johnston, J. / Dumbill, E.: Programming web services with XML-RPC: Creating web application gateways, Beijing [u.a.] 2001.

Schoder / Fischbach / Teichmann (2002): Schoder, D. / Fischbach, K. / Teichmann, R.: Peer-to-peer: ökonomische, technologische und juristische Perspektiven, Berlin [u.a.] 2002.

Scribner / Stiver (2000): Scribner, K. / Stiver, M. C.: Understanding SOAP, Indianapolis 2000.

Seely (2002): Seely, S.: SOAP: cross platform Web service development using XML, Upper Saddle River, NJ 2002.

Siegel (1999): Siegel, J. (Hrsg.): An overview of CORBA 3, Helsinki 1999.

Simon E. (1996): Simon E.: Distributed Information Systems - From Client/Server to Distributed Multimdia, London 1996.

Skonnard (2002): Skonnard, A.: XML Files - Publishing and discovering Web Services with DISCO and UDDI.. In: Microsoft systems journal (2002) S. 23-34,

Sleeper / Robins (2002): Sleeper, B. / Robins, B.: The Law of Evolution: A Pragmatic Analysis of the Emerging Web Services Market, San Francisco 2002.

Snell / Tidwell / Kulchenko (2002): Snell, J. / Tidwell, D. / Kulchenko, P.: Programming web services with SOAP, Beijing [u.a.] 2002.

Steckermeier (2001): Steckermeier, M.: Virtuelle, private Rechner: eine Software-Architektur für verteilte Anwendungen, Erlangen 2001.

Stevens (2002a): Stevens, M., Service Oriented Architecture Part 1, http://www.developer.com/db/print.php/10920_1010451_2, Abruf am 24.05.2003. (a)

Stevens (2002b): Stevens, M., Service Oriented Architecture Part II, http://www.developer.com/db/print.php/1014371, Abruf am 24.05.2003. (b)

Thompson (2002): Thompson, H., The Extensible Stylesheet Language (XSL), http://www.w3.org/Style/XSL/, Abruf am 30.04.2003.

Thompson / Beech / Maloney / Mendelsohn (2001): Thompson, H. S. / Beech, D. / Maloney, M. / Mendelsohn, N., XML Schema Part 1: Structures, http://www.w3.org/TR/xmlschema-1/, Abruf am 30.04.2003.

Wall / Lader (c2002): Wall, L. / Lader, A.: Building Web services and .NET applications, New York, NY [u.a.] c2002.

WebServices.Org (2003): WebServices.Org, The Web Service Community Portal, http://www.webservices.org,

Weitzel / Harder / Buxmann (2001): Weitzel, T. / Harder, T. / Buxmann, P.: Electronic Business und EDI mit XML, Heidelberg 2001.

Wojciechoski / Weinhardt (2002): Wojciechoski, R. / Weinhardt, C.: Web Services und Peer-to-Peer-Netzwerke, In: Schoder, D. / Fischbach, K. / Teichmann, R.: Peer-to-Peer - Ökonomische, technologische und juristische Perspektiven, Berlin [u.a.] 2002, S. 99-117.

Wood (1998): Wood, L., Document Object Model (DOM) Level 1 Specification, http://www.w3.org/TR/REC-DOM-Level-1/, Abruf am 30.04.2003.

o. V. (2003): o. V., Web Services Journal, http://www.sys-con.com/webservices, Abruf am 01.09.2003.