Technische Universität München

Forschung- und Lehreinheit Informatik III
Prof. R. Bayer Ph.D., Prof. Dr. D. Kossmann

Hauptseminar Informatik im Sommersemester 2003

Web Services

W3C Standards I:
SOAP, WSDL und UDDI

Referentin: Tania Fichtner Brunswig

Betreuer: Dipl.-Inf. Steffen Rost

Abgabetermin: 17.April 2003

Votragsdatum: 24. April 2003

Inhaltsangabe

1. Einführung

Web Services können als Vermittler zwischen Client- und Server-Applikationen gesehen werden. Die Aufgaben eines Web Services sind somit klar definiert:

1. Annehmen von Anfragen seitens des Clients
2. Durchreichen dieser Informationen an eine Serverkomponente
3. Abwarten bis ein Ergebnis zurückgeliefert wird
4. Weiterleiten des Rückgabewertes an den Client.

Diese „Vermittler"-Funktionalität ist nicht neu. Bisher wurde die Kommunikation zwischen Rechnern in verteilten Systemen durch den Einsatz von Middleware-Technologien ermöglicht. Eine bekannte Middleware Technologie im B2B-Bereich ist CORBA.
Das zentrale Organ der Kommunikationsübertragung ist ein so genannter "Broker" (Vermittler). Ein Broker nimmt Anfragen entgegen, leitet sie an eine Serverkomponente weiter und liefert dem anfragenden Client einen entsprechenden Rückgabewert zurück.

1.1 Web Service Architektur

Man kann die Web Service Architektur auf zwei verschidene Arten beschreiben. Eine davon ist, die einzelnen Rollen der Web Service Akteure zu untersuchen; und die andere, den „Web Service Protocol stack" zu betrachten.

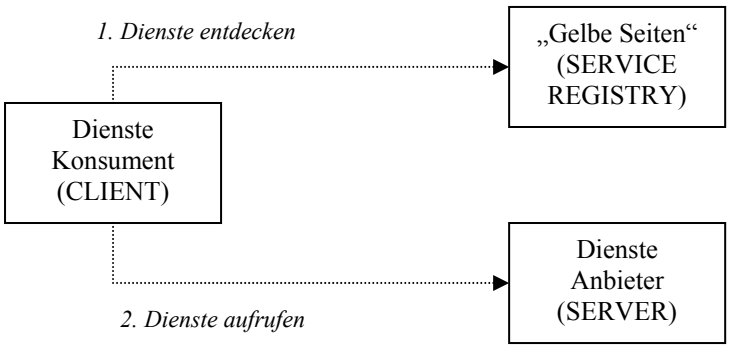
Web Service Rollen

Insgesamt lassen sich drei Rollen vergeben: der Dienst-Anbieter (Server), der Dienst-Konsument (Client) und die „Gelben Seiten" (Service Registry).

- Der Dienste Anbieter entwickelt eine Dienstleistung, und stellt sie über das Internet zur Verfügung.
- Der Dienste Konsument kann irgendjemand sein. Der Konsument nutzt ein Web Service indem er eine Verbindung aufbaut und eine XML Anfrage abschickt.
- Die „Gelben Seiten" sind eine standardisierte, zentrale Einheit, wo Web Services aufgelistet werden. Dienste Anbieter können hier ihre Web Services veröffentlichen.

[ECER02].

Abbildung1: Web Service Rollen



Web Service Proocol Stack

Der Web Service Protocol Stack ist eine Sammlung von Protokollen, die zum definieren, finden und implementieren von Web Services gedacht ist.
Bisher enthält dieser Stack vier Schichten:

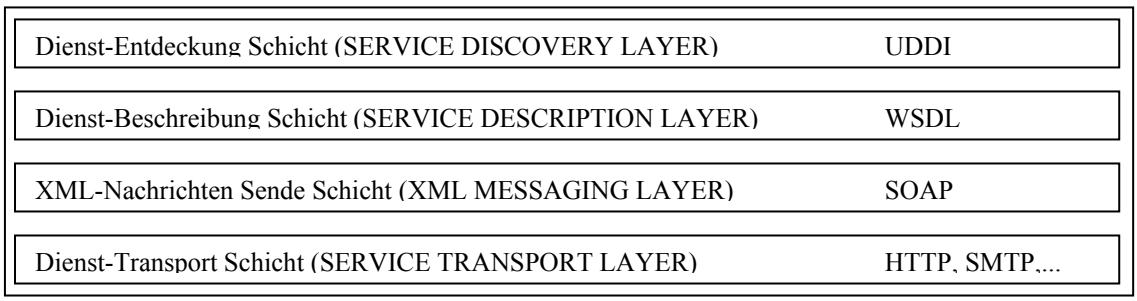
Dienst-Entdeckung Schicht (Service Discovery Layer): Diese Schicht kümmert sich um die zentrale Führung einer Dienste Liste.

Dienst-Beschreibung Schicht (Service Description Layer): Die Verantwortung dieser Schicht ist die Schnittstellen der Dienste einheitlich zu beschreiben.

XML-Nachrichten Sende Schicht (XML Messaging Layer): Diese Schicht ist verantwortlich, die Nachrichten in ein allgemein verständliches XML Format umzuwandeln. Somit können sich die Rollen auf einer gemeinsamen Basis verständigen. Diese Schicht wird durch das SOAP Protokoll repräsentiert.

Dienst Transport Schicht (Service transport laye)r: Diese Schicht kümmert sich um den Transport der Nachrichten die zwischen den Applikationen gesendet werden. Momentan sind folgende Protokolle Teil dieser Schicht: HTTP, SMTP, FTP, und neue Protokolle, so wie das Blocks Extensible Exchange Protocol (BEEP).
[ECER02]

Abbildung2: Web Service Protocol Stack



1.2 Ziele

Um ein konkretes Bild zu bekommen was Web Services sind und können, ist es wichtig die Technologien zu verstehen die sich dahinter verbergen. Dazu werden in den nächsten Kapiteln diese Technologien (SOAP, WSDL, UDDI) näher betrachtet.

2. SOAP

Das **S**imple **O**bject **A**ccess **P**rotocol legt eine Formatierung fest, mit der die Informationen, die von einem Rechner zum anderen übertragen werden, codiert werden müssen, um eine „globale“ Verständigung zwischen den beteiligten Rechnern zu erreichen. SOAP basiert auf der XML-Syntax und kann daher von XML-Parsern eingelesen werden, ohne dass hierfür erst noch eigene Lösungen programmiert werden müssen.

Im Umfeld der Web Services kann SOAP als „Verpackungseinheit“ definiert werden. Die bei Web Services ausgetauschten Informationen müssen so verpackt werden, dass jeder beteiligte Rechner diese Informationen lesen und verstehen kann. [EN02]

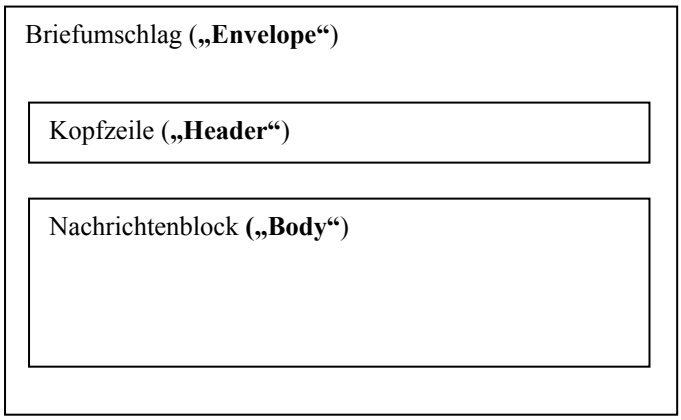
SOAP umfasst nur die Formatierung von Daten, nicht aber, wie die Daten von A nach B gelangen oder wie es technisch realisierbar ist, dass ein entfernter Methodenaufruf (RPC: Remote Procedure Call) über das Internet durchgeführt wird. Das heißt, dass die Art wie ein RPC auf dem Zielrechner ausgeführt wird, Sache des Dienst Anbieters ist. Der Dienste Konsument muss wissen, wo er den Dienst wie aufruft.

Eine gemäß dem SOAP-Format kodierte Information wird als SOAP-Nachricht bezeichnet. Eine SOAP-Nachricht ist im Grunde genommen eine Ein-Weg-Kommunikation zwischen SOAP-Knoten, von einem SOAP-Sender zu einem SOAP-Empfänger. Diese einfachen SOAP-Nachrichten werden in den Anwendungen zu komplexeren Interaktions-Mustern kombiniert (Request/Response, Konversationen, usw.).
Es ist möglich, dass eine SOAP-Nachricht durch ein oder mehrere SOAP-intermediaries gesendet wird, bevor sie den SOAP-Empfänger erreicht. Diese intermediaries können Einfluss auf die Nachricht haben. [S002]

2.1 Aufbau einer SOAP Nachricht

Eine SOAP-Nachricht kann mit einem normalen Brief verglichen werden. Denn genau wie ein Brief besitzt eine SOAP-Nachricht einen Umschlag (*envelope*), einen optionalen Briefkopf (*header*) und den eigentlichen Inhalt (*body*). Der Umschlag dient als „Verpackung“ des Inhalts und bildet das Wurzelelement des XML-Dokuments (siehe Abbildung x).

Abbildung 3: Analogie zwischen einer SOAP-Nachricht und einem Brief



Das XML-Dokument in Beispiel 1 beinhaltet eine SOAP-Nachricht, die einen entfernten Methodenaufruf (RPC) bewirken soll.

Beispiel 1: SOAP Nachricht

```
1     <?xml version='1.0' ?>
2     <!--            Briefumschlag: ENVELOPE                 -->
3     <env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
4       <!--          Kopfzeile: HEADER          -->
5       <env:Header>
6         <xyz:transaction xmlns:xyz="soap-transaction"
7              env:mustUnderstand="true">
8              <transactionID>1234</transactionID>
9         </xyz:transaction>
10      </env:Header>
11     <!--           Nachrichtentext: BODY          -->
12     <env:Body>
13      <dienst:getKurs xmlns:dienst="urn:RealTimeKursService">
14        <wkn xsi:type="xsd:string">923835</wkn>
15      </dienst:getKurs>
16     </env:Body>
17    <!--            Ende            -->
18    </env:Envelope>
```

Im Nachfolgenden werden die einzelnen Elemente der SOAP-Nachricht genauer erklärt.

2.2 Wie verpackt SOAP die Informationen?

2.2.1 SOAP Envelope

Der `Envelope` repräsentiert das Wurzelelement einer SOAP-Nachricht. Er umschließt den optionalen `Header` und den unbedingt anzugebenden `Body (Beispiel 2)`:

Beispiel 2: Envelope-Element

```
1     <env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
2       <env:Header>
3         ...
4       </env:Header>
5       <env:Body>
6         ...
7      </env:Body>
8     </env:Envelope>
```

Der Namespace `env` ist eine Abkürzung für SOAP-Envelope und signalisiert, dass die dazugehörigen Elemente (wie z.B. `env:Body`) ein Bestandteil des Umschlags sind. Der Namespace `env` zeigt auf den URI `http://www.w3.org/2002/12/soap-envelope`. Hinter dieser URL verbirgt sich ein XML Schema, welches den Aufbau der SOAP-Nachricht festlegt. Für den in Beispiel 2 verwendeten `env`-Namespace (Bsp. 2, Zeile 1) wird in Beispiel 3 ein Auszug des dazugehörigen XML Schemas gezeigt. [S002]

Beispiel 3: Auszug aus XML Schema für env

```
1     ...
2     <xs:element name="Envelope" type="tns:Envelope" >
3
4     <xs:complexType name="Envelope">
5       <xs:sequence>
6         <xs:element ref="tns:Header" minOccurs="0"/>
7         <xs:element ref="tns:Body" minOccurs="1"/>
8         <xs:any namespace="##other"
9                   minOccurs="0"
10                  maxOccurs="unbounded"
11                  processContents="lax"/>
12      </xs:sequence>
13    ...
```

Eventuell auftretende Fehler auf Serverseite werden durch das `Fault`-Element beschrieben. Zum Beispiel wenn eine SOAP-Nachricht von Clientseite versendet wird und diese nicht den gleichen Namespace wie die Serverseite angehört, sendet der Server einen Versionskonflikt, der mit Hilfe eines `Fault`-Elements formatiert wird. Näheres zum `Fault`-Element wird im übernächsten Abschnitt (2.2.3) besprochen.

2.2.2 SOAP Header

Der optionale Header einer SOAP-Nachricht bietet die Möglichkeit, bestimmte Informationen wie zum Beispiel Transaktionsdaten, Authentisierungsdaten oder Sicherheitsdaten, in dem zu übertragenden XML-Dokument unterzubringen. Hiermit sind solche Informationen in einer SOAP-Nachricht untergebracht, aber die Art und Weise dieser Unterbringung ist keinesfalls standardisiert. Dass heißt, dass SOAP-Nachrichten Zugangsinformationen beinhalten können und der Empfänger daher jedes Mal genau darüber informiert sein muss, wo diese Daten „versteckt“ sind. Deshalb muss einer der beiden Kommunikationspartner festlegen, wie Zugangsinformationen in das XML-Dokument integriert werden müssen.

Im nächsten Beispiel wird eine SOAP-Nachricht mit mehreren Header Blöcken gezeigt.

Beispiel 4: Zugangsdaten im Header einer SOAP-Nachricht

```
1     <?xml version="1.0" ?>
2      <env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
3        <env:Header>
4          <login:sicherheit xmlns:login="http://example.com"
5                 env:role=http://example.com/Login
6                  env:mustUnderstand="true">
7            <login:benutzername>Peter Muster</login:benutzername>
8            <login:passwort>meinpasswort</login:passwort>
9          </login:sicherheit>
10         <audit:administration xmlns:audit="http://example.com/audit"
11         env:role="http://www.w3.org/2002/12/soap-envelope/role/next">
12         ...
13         ...
14         </audit:administration>
15       </env:Header>
16       <env:Body >
17         ...
18         ...
19       </env:Body>
20    </env:Envelope>
```

Für die Elemente im `Header` (Bsp. 4, Zeile3)gibt es die Vorgabe, maximal drei Attribute besitzen zu dürfen: `mustUnderstand` `role` und `relay`. Das Attribut `role` (Bsp. 4, Zeile5) steht für die Komponente (SOAP-Knoten), die sich um die Auswertung der `Header`-Information kümmern soll. Das mustUnderstand (Bsp. 4, Zeile 6) Attribut kann benutzt werden, um den Empfänger der SOAP-Nachricht mitzuteilen, dass das zugehörige `Header`-Element auf jeden Fall von Ihm verstanden werden muss. Falls der Empfänger diese Nachricht nicht verarbeiten kann, muss er sie ablehnen. Das relay Attribut zeigt an, ob ein Header Blcok “relayed“ werden muss, wenn dieser Header Block nicht vom angesteuerten SOAP-Intermediary verarbeitet wird. [S202]

2.2.3 SOAP Body

Innerhalb des `Body`-Elements werden die Informationen beschrieben, die die Empfängerseite erreichen sollen, damit dort ein Verarbeitungsprozess in Gang gesetzt werden kann. Zum Beispiel, enthält das `Body`-Element entfernte Methoden Aufrufe und Antworten (RPC requests

and responses). Ein Beispiel für ein RPC request im Body Element wurde bereits in Beispiel 1 gezeigt. Hier wurde die Methode getKurs (Bsp. 1, Zeile 13) aufgerufen, und das Parameter wkn mit dem String-Wert “923835“ übergeben.
Eine typisches RPC response könnte wie in Beispiel 5 aussehen:

Beispiel 5: RPC response mit einem output Parameter für den Aufruf aus Beispiel 1

```
1     <?xml version='1.0' ?>
2
3     <env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
4
5       <env:Header>
6         <xyz:transaction xmlns:xyz=“soap-transaction“
7              env:mustUnderstand=“true“>
8              <transactionID>1234</transactionID>
9         </xyz:transaction>
10      </env:Header>
11
12     <env:Body>
13      <dienst:getKursResponse xmlns:dienst=”urn:RealTimeKursService”>
14        <value xsi:type=“xsd:double“>45.9</value>
15      </dienst:getKursResponse>
16     </env:Body>
17
18    </env:Envelope>
```

Die Antwort ist im Body-Element enthalten, in Form eines getKursResponse-Elements. Dieser Element-Name enthält den ursprünglichen Methodenaufrufnamen “getKurs“ plus das nach Konvention angehängte Wort “Response“. Das output Parameter “value“ enthält den Wert den die aufgerufene Methode zurückliefert. In diesem Fall den Kurswert der angeforderten Aktie.

Wie schon vorher angesprochen ist ein weiterer Inhalt des `Body`-Elements die Übertragung von Fehlern.

SOAP Fehler
Ein SOAP-Fehler ist eine spezielle Nachricht, die dem anfragenden Programm mitteilt, welcher Fehler aus welchem Grund aufgetreten ist, während versucht wurde, die SOAP-Nachricht des Anfragers zu bearbeiten. Ein Beispiel einer SOAP-Fehler-Nachricht kann in Beispiel 6 gesehen werden.

Die Fehler werden durch das `Fault`-Element (Bsp. 6, Zeile 5) beschrieben. Das `Fault`-Element enthält zwei obligatorische Sub-Elemente, `env:Code` (Bsp. 6, Zeile 6) und `env:Reason` (Bsp. 6, Zeile 12), und optional das Anwendungs-spezifische Sub-Element `env:Detail` (Bsp. 6, Zeile 17). Weitere optionale Sub-Elemente sind `env:Node und env:Role`. `env:Node` definiert welcher SOAP-Sender den Fehler generiert hat, und `env:Role` spezifiziert die Rolle dieses Senders.

Das `env:Code` Sub-Element besteht selbst aus den Sub-Elementen `env:Value` (obligatorisch) und `env:Subcode` (optional).Es ist ein algorithmisch generierter Fehlercode, dessen Wert dem XML-Zeichensatz gerecht wird. Der env:Subcode (Bsp. 6, Zeile 8), wenn angezeigt dient dazu, den Fehler genauer zu spezifizieren.
Die Struktur des env:Subcode Elements ist hierarchisch; jedes Kind hat wiederum die Sub-Elemente `env:Value` (obligatorisch) und `env:Subcode` (optional). Diese hierarchische Struktur des env:Code Elements erlaubt es Fehler hierarchisch anzuordnen.

Beispiel 6: Fehlerinformationen in einer SOAP-Nachricht.

```
1     <?xml version='1.0' ?>
2     <env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope"
3                   ...>
4       <env:Body>
5        <env:Fault>
6          <env:Code>
7            <env:Value>Client.Authentication</env:Value>
8            <env:Subcode>
9             <env:Value>rpc:BadArguments</env:Value>
10           </env:Subcode>
11         </env:Code>
12         <env:Reason>
13          <env:Text xml:lang="de">Login-Daten sind ungültig</env:Text>
14          <env:text xml:lang="en-US">Login Data is not valid</env:Text>
15         </env:Reason>
16         <!—programmspezifische Fehlermeldung -->
17         <env:Detail>
18          <e:myFaultDetails
19            xmlns:e="http://www.example.org/faults">
20            <e:message>Name does not match logindata</e:message>
21            <e:errorcode>999</e:errorcode>
22          </e:myFaultDetails>
23         </env:Detail>
24       </env:Fault>
25     </env:Body>
26    </env:Envelope>
```

Das `env:Reason` Sub-Element ist eine für Menschen lesbare Fehlermeldung. Es muss mindestens ein `env:Text` (Bsp. 6, Zeile 13) Element enthalten. Jedes `env:Text` Element muss mit einem eindeutigen `xml:lang` (Bsp. 6, Zeile 13) Attribut versehen werden. Somit können die Fehlermeldungen in mehreren Sprachen zur Verfügung stehen.[S102]

Es kann auch passieren dass ein Fehler generiert wird, weil ein Element im Header nicht verstanden wurde. Solche Fehler werden auch mit dem `env:Fault` Element im `env:Body` signalisiert, jedoch zusätzlich im jeweiligen Header mit einem speziellen Element `env:NotUnderstood` hervorgehoben.
In Beispiel 7 wird eine SOAP Nachricht gezeigt, die einen Fehler beim einlesen des Headers (Bsp. 7, Zeile 4) verursacht hat.

Beispiel 7: SOAP Nachricht mit Fehler Element

```
1     <?xml version='1.0' ?>
2     <env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope>
3      <env:Header>
4         <env:NotUnderstood qname="login:sicherheit"
5               login:sicherheit xmlns:login="http://example.com"/>
6      </env:Header>
7      <env:Body>
8       <env:Fault>
9        <env:Code>
10        <env:Value>env:MustUnderstand</env:Value>
11       </env:Code>
12       <env:Reason>
13        <env:Text xml:lang="en-US"> Header nicht verstanden</env:Text>
14          <env:Text xml:lang="en-US"> Header not understood </env:Text>
15       </env:Reason>
16      </env:Fault>
17     </env:Body>
```

```
18    </env:Envelope>
```

Wenn mehrere Header Blöcke nicht verstanden werden, dann wird jeder Block einzeln identifiziert und in einer Serie von `env:NotUnderstood` Elementen aufgelistet.

2.3 Die Datencodierung von SOAP

Da SOAP auf XML basiert, können in SOAP auch die Gestaltungsmöglichkeiten von XML ausgenutzt werden. Solange Elemente den Namensraum von `env` verwenden, orientieren sich der Aufbau und die Struktur der Elemente gemäß den folgenden Erklärungen. Es kann aber auch ein selbst definierter Elementenaufbau gewählt werden, dem ein anderes XML Schema zugrunde liegen kann. Dafür steht das Attribut `env:ecodingStyle` zur Verfügung, um `Header` Blöcke oder `Body` Sub-Elemente speziell zu qualifizieren.

Einfache Datentypen

Einfache Datentypen sind in Anhang A aufgelistet. Dabei handelt es sich um atomare Datentypen, die sich nicht aus anderen Typen zusammensetzen, wie z.B. ein Struct.

Beispiel 8: Ein einfacher Datentyp xsd:string.

```
1     <?xml version='1.0' ?>
2     <env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope"
3        env_ENC:encodingstyle="http://www.w3.org/2002/12/soap-encoding"
4        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6        xmlns :env-ENC="http://www.w3.org/2002/12/soap-encoding">
7       <env:Body>
8       <getKurs>
9        <wkn xsi:type=”xsd:string”>923835</wkn>
10      </getKurs>
11     </env:Body>
12    </env:Envelope>
```

Zusammengesetzte Datentypen

In SOAP werden zwei zusammengesetzte Datentypen definiert: `struct` und `array`. Ein `struct` ist eine Zusammensetzung von Daten unterschiedlichen Typs. Ein `array` im Gegensatz dazu ist eine Liste von Daten gleichen Typs.

Structs [S202]

Ein `struct` ist meistens die Zusammensetzung mehrerer atomarer Datentypen zu einem Hochtyp, z.B. die `Adresse`, die sich aus `Name`, `Strasse`, `Hausnummer`, `PLZ` und `Wohnort` zusammensetzt. Eine XML-Instanz dieses Structs ist im Besipiel 9 zu sehen.

Beispiel 9: Hochtyp Adresse in SOAP

```
1     <ns1:Adresse>
2       <Name>Peter Muster</Name>
3       <Strasse>Hauptstrasse</Strasse>
4       <Hausnummer>13</Hausnummer>
5       <PLZ>12345</PLZ>
6       <Wohnort>München</Wohnort>
7     </ns1:Adresse>
```

Es ist auch möglich Verweise unterzubringen. Dabei verweisen eine oder mehrere Referenzen auf mit Primärschlüsseln ausgestattete Elemente in einem Dokument.

Arrays

Anwendungen können als Rückgabewerte, oder Anfrageparametern Arrays enthalten. Diese müssen in ein SOAP-konformes Array umgewandelt werden.

Beispiel 10: SOAP-konformes Array of String

```
<getKursResponse
   xsi:type="env-ENC:Array"
   enc-ENC:arrayType="xsd:string[3]"
   xmlns:env-ENC=...
>
  <item>923853</ item >
  < item >Apple</ item >
  < item >45.3</ item >
</getKursResponse>
```

Hier wird der Inhalt des `getKursResponse`-Elements auf den xsi-Typ `SOAP-ENC:Array` gesetzt. Mit dieser Typisierung wurde festgelegt, dass der Inhalt ein Array ist. Damit auch spezifiziert werden kann, was für ein Array folgt, wird anschließend der `SOAP-ENC:arrayType` mit dem dreifachen Array of String `xsd:string[3]` belegt.

Man kann auch selber Typdefinitionen durchführen. Zum Beispiel (Bsp: 11) wenn man einen Integertyp spezifizieren möchte, der eine Null voranhängt, falls der Wert kleiner 10 ist.

Beispiel 11: Code um eigenen Integertyp zu spezifizieren

```
<element name="Aktennummern" type="env-ENC:Array"/>
<simpleType name="meineTypdefinition">
  <restriction base="xsd:int">
   <pattern value="0\[0-9]">
  </restriction>
</simpleType>
...
<Aktennummern env-ENC:arrayType="xsd:int[3]">
  <nummer xsi:type="meineTypdefinition">00</nummer>
  <nummer xsi:type="meineTypdefinition">01</nummer>
  <nummer xsi:type="meineTypdefinition">02</nummer>
</Aktennummern>
```

Der Arraytyp muss nicht unbedingt ein Simpletyp sein. Ein Struct als Arraytyp kann genauso gebildet werden. In Beispiel 12 wird so ein Array gezeigt, mit dem aus dem in Beispiel 9 definierten Struct "ns1:Adresse".

Beispiel 12: Ein Struct als Arraytyp.

```
<AdresseArray env:arrayType="ns1:Adresse[2]">

<ns1:Adresse>
  <Name>Peter Muster</Name>
  <Strasse>Hauptstrasse</Strasse>
  <Hausnummer>13</Hausnummer>
  <PLZ>12345</PLZ>
  <Wohnort>München</Wohnort>
</ns1:Adresse>

<ns1:Adresse>
  <Name>Peter Muster</Name>
  <Strasse>Hauptstrasse</Strasse>
  <Hausnummer>13</Hausnummer>
  <PLZ>12345</PLZ>
  <Wohnort>München</Wohnort>
</ns1:Adresse>
```

```
18
19    </AdresseArray>
```

Es ist sogar möglich ein Array vom Typ Array zu bilden. Hierfür kann der Inhalt durch Nutzung von Querverweisen gebildet werden.

Mehrdimensionale Arrays werden so wie eine Tabelle abgetragen. Das nächste Beispiel zeigt ein 2x3-dimensionales Array of String:

Beispiel 13: Mehrdimensionales Array.

```
1     <Beispiel env-ENC:arrayType="xsd:string[2,3]">
2       <item>Element 1-1</item>
3       <item>Element 1-2</item>
4       <item>Element 2-1</item>
5       <item>Element 2-2</item>
6       <item>Element 3-2</item>
7       <item>Element 3-2</item>
8     </Beispiel>
```

2.4 Zusammenfassung

Dieses Kapitel hat gezeigt, dass die Daten, die zwischen A und B gesendet werden, mittels XML formatiert wurden. Dieses Format wurde als SOAP bezeichnet.
Eine sehr starke Eigenschaft von SOAP ist, dass es möglich ist, die Datentypisierung jeder Programmiersprache darzustellen.
Im nächsten Kapitel geht es mit der Erläuterung einer weiteren Schicht des Web Service Protocol Stacks weiter: Dienst-Beschreibung Schicht mittels WSDL.

3. WSDL

WSDL steht für Web Services Description Language und ist eine auf XML basierende Sprache. Mit Hilfe von WSDL kann definiert werden, welche Methoden bei der Serverkomponente vom Client ausgeführt werden können, welche Parameter dabei übergeben werden müssen und was für einen Rückgabewert diese einzelnen Methoden liefern.

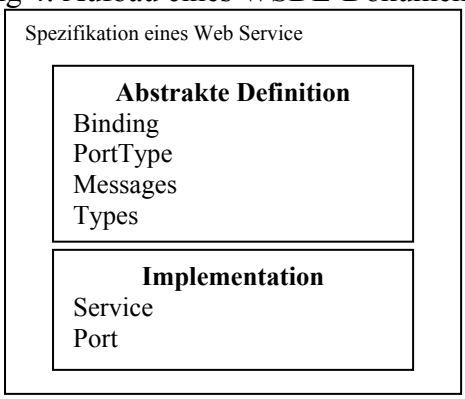
Um Web Services verstehen zu können, kann man sie, wie in der Einführung kurz angesprochen mit der bereits existierenden Middleware Technologie vergleichen.
Zum Beispiel CORBA: Um Methoden seitens des Clients auf dem Server ausführen zu können, müssen die hierfür nötigen Informationen vom Client an den Broker gesendet werden, der erkennt, welche Methode auf welchem Server ausgeführt werden soll. Um die Programmierung zu vereinfachen, werden die aufrufbaren Methoden des Servers in Form eines Interface publiziert. Dieses Interface ist mit der IDL (CORBAs Interface Description Language) entwickelt. Nach der Publikation kann sich jeder Programmierer, dessen Programmiersprache CORBA-fähig ist, aus diesem Interface Programmcode für seine entsprechende Programmiersprache generieren lassen. Das bedeutet dass nach der IDL-Publikation es egal ist, mit welcher Programmiersprache der Server entwickelt wurde. Es kommt nur noch darauf an, ob der Client mit seiner Programmiersprache auf den CORBA-Server zugreifen kann. [ECER02]

Dieses Prinzip ist auch auf Web Services anwendbar, denn was die IDL für CORBA ist, ist WSDL für Web Services. Die Besonderheit eines WSDL-Dokuments ist die Angabe unter welcher Adresse (URL) der Web Service zu finden ist. Diese Information wird über eine Service Registry (UDDI, Kapitel 4) zur Verfügung gestellt, so dass IP-Adressänderungen nicht mehr allen Clients mitgeteilt werden müssen.

WSDL entstand auf Initiative der Unternehmen IBM, Microsoft und ARIBA. Mittlerweile wurde die Technologie vom W3C standardisiert.

Abbildung 4 zeigt den Aufbau einer Web Service Beschreibung. Ein WSDL Dokument wird in zwei Teile aufgeteilt. Zum einen existieren eine abstrakte und wieder verwendbare Definition im oberen Teil und zum anderen eine implementierungsabhängige Definition im unteren Teil des Dokuments. Die erste ist unabhängig von dem jeweils benutzten Transportprotokoll. Das tatsächlich verwendete Transportprotokoll (SOAP, http etc.) wird innerhalb eines Binding-Elements untergebracht. Dadurch wird der Web Service als solcher technologieunabhängig.

Abbildung 4: Aufbau eines WSDL-Dokuments für einen Web Service



3.1 WSDL-Dokument

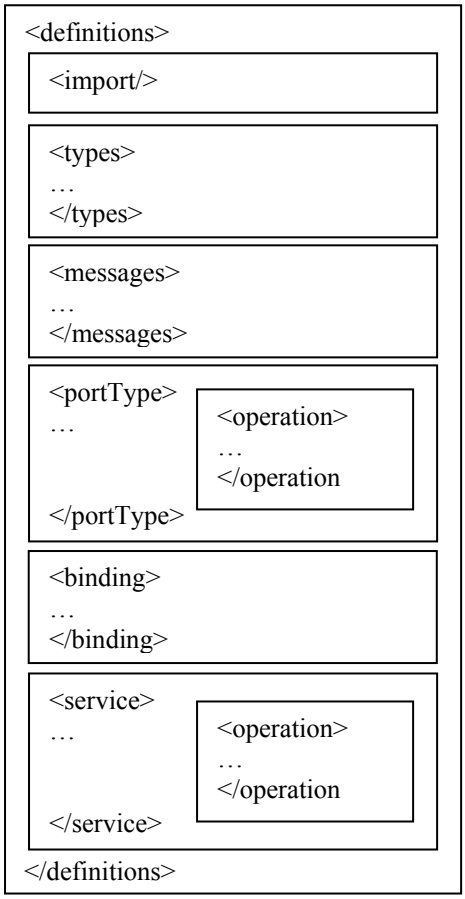
Um ein Web Service definieren zu können, muss ein WSDL-Dokument mehrere Möglichkeiten bieten, um flexibel auf jeden einzelnen Web Service zu reagieren. Daher wird ein WSDL-Dokument in sechs Teile aufgeteilt:

1. Die Adresse des Web Service. Die Adresse eines Web Service bildet zusammen mit den Nachfolgenden Elementen des WSDL-Dokuments den so genannten `service`.
2. Die Datentypen der einzelnen Rückgabe- und Übergabewerte: `types`.
3. Die Eingabe- und Ausgabeparameter, die `message` genannt werden.
4. Die Methodensignaturen, die mit `operation` bezeichnet werden.
5. Die Zuordnung der einzelnen Methoden zu einer Operation eines Web Service : `portType`.
6. Das Trägerprotokoll, über das kommuniziert wird, wie z.B. http. Dies wird mit `binding` bezeichnet.

Die Struktur eines WSDL-Dokuments ist in Abbildung 5 dargestellt. Beispiel 14 zeigt ein WSDL-Dokument für ein RealtimeKursService- Service Beispiel.

Aus welchen Elemente ein WSDL-Dokument besteht, wird nun im Detail beschrieben.

Abbildung 5: Skizze eines WSDL-Dokuments.



Beispiel 14: WSDL-Dokument des RealtimeKursService.

```
1     <wsdl:definitions
2       targetnamespace=”“
3       xmlns=”wsdlsoap=”
4       xmlns:xsd=”“
5       xmlns:tns1=”“
6       xmlns:env-ENC=”“
7       xmlns:intf=”“
8       xmlns:wsdl=”“
9       xmlns:impl=”“
10      xmlns=”“>
11
12    <types>
13      <schema xmlns=http://www.w3.org/2001/XMLSchema
14       targetNamespace=“RealtimeKursService“>
15        <complexType name=“Aktienkurs“>
16          <sequence>
17            <element name=”name” nillable=”true” type=”xsd:string” />
18            <element name=”wkn” nillable=”true” type=”xsd:string” />
19            <element name=”kurs” type=”xsd:double” />
20          </sequence>
21        </complexType>
22      <element name=”Aktienkurs” nillable=”true” type=”tns1:Aktienkurs”
    />
23      </schema>
24    </types>
25
26    <wsdl:message name=”getKursResponse”>
27      <wsdl:part name=”return” type=”tns1:Aktienkurs” />
28    </wsdl:message>
29    <wsdl:message name=”getKursRequest”>
30      <wsdl:part name=”wkn” type=”xsd:string” />
31    </wsdl:message>
32
33    <wsdl:portType name=”RealtimeKursService”>
34      <wsdl:operation name=”getKurs” parameterOrder=”wkn”>
35        <wsdl:input message=”intf:getKursRequest” />
36        <wsdl:output message=”intf:getKursResponse” />
37      </wsdl:operation>
38    </wsdl:portType>
39
40  <wsdl:binding name=”RealtimekursService-2SoapBinding”
    type=”intf:RealtimeKursService”>
41        <wsdlsoap:binding style=”rpc” transport=”.../http” />
42      <wsdl:operation name=”getKurs”>
43      <wsdlsoap:operation soapAction=”“ />
44      <wsdl:input>
45        <wsdlsoap:body use=”encoded”
46          encodingStyle=”.../encoding”
47          namespace=”getKurs”
48        />
49      </wsdl:input>
50      <wsdl:output>
51        <wsdlsoap:body unse=”encoded” encodingStyle=http://.../encoding
           namespace=”.../services/RealtimeKursService” />
52      </wsdl:output>
53      </wsdl:operation>
54    </wsdl:binding>
55
56    <wsdl:service name=”RealtimeKursService_Service”>
57      <wsdl:port name=”RealtimeKursService”
```

```
58        binding="intf:RealtimeKursService-2SoapBinding">
59       <wsdlsoap:address
    location=http://localhost:8080/axis/services/RealtimeKursService />
60       </wsdl:port>
61     </wsdl:service>
62
63     </wsdl:definitions>
```

3.2 Elemente eines WSDL-Dokuments

[WSDL03], [WSDR02], [ECER02].

3.2.1 Das `<definitions>` Element

Das `definitions` Element bildet das Wurzelelement eines WSDL-Dokuments. Es umschließt daher alle nachfolgenden Elemente und bietet eine gute Möglichkeit, die benutzten Namespaces zu definieren.

Beispiel 15: Typische Attributierung für ein WSDL-Dokument

```
<wsdl:definitions
  targetnamespace=""
  xmlns="wsdlsoap="
  xmlns:xsd=""
  xmlns:tns1=""
  xmlns:env-ENC=""
  xmlns:intf=""
  xmlns:wsdl=""
  xmlns:impl=""
  xmlns=""
>
```

Die Benutzung von Namespaces ist wichtig um die einzelnen Elemente unterscheiden zu können. Das `definitions Element` spezifiziert auch das targetNamespace Attribut. Dieses Attribut ist eine Konvention aus dem XML Schema, dass dem WSDL Dokument erlaubt, sich auf sich selbst zu beziehen.

3.2.2 Das `<import>` Element

Ein XML Dokument muss sich nicht unbedingt aus einer physischen Datei zusammensetzen. XML wurde für verteilte Umgebungen entwickelt und kann sich daher aus vielen Dokumententeilen zusammensetzen.

Da WSDL auf XML basiert, ist es auch möglich, die Teile des Dokuments in verschiedenen Dateien an verschiedenen Orten abzulegen. Dies ist sinnvoll, wenn der zu beschreibende Web Service mit bereits existierenden Services zusammenarbeitet. Wenn sich einer dieser Services ändert, muss nur ein Einzelteil der gesamten Service-Beschreibung geändert werden.

Dazu ist das `import`-Element zuständig: für die Einbindung von externen WSDL-Ressourcen.

3.2.3 Das `<types>` Element

Das `types`-Element ist eine Position im WSDL-Dokument, an der die Möglichkeit besteht, alle die Datentypen zu definieren, die nicht vom XML Schema-Standard erfasst werden. Dieses sind sämtliche Hochtypen, die mittels `complexType` selbst definiert werden müssen. Zum Beispiel könnte das Service RealtimeKursService in Java programmiert werden, und als Rückgabewert der Methode getKurs ein Objekt der Klasse Aktienkurs enthalten. Dieses Aktienkurs-Objekt müsste mittels complexType selbst definiert werden.

Wenn also mehrere Datentypen innerhalb eines WSDL-Dokuments definiert werden, bedeutet das für die Nicht-Java-Sprachen, die auf den mit Java programmierten Web Service zugreifen, dass sie auf jeden Fall sämtliche Übergabe- und Rückgabeparameter erzeugen und verarbeiteen können. Innerhalb des `types`-Elements werden nur die Datentypen definiert, die von den Methoden des Web Service entweder an den Client zurückgegeben werden oder die der Client einer dieser Methoden übergeben muss.

Beispiel 16: Das types-Element des im RealtimeKursService verwendeten Adresse-Objekts.

```
1     <types>
2       <schema xmlns=http://www.w3.org/2001/XMLSchema
3        targetNamespace="RealtimeKursService">
4         <complexType name="Aktienkurs">
5           <sequence>
6             <element name="name" nillable="true" type="xsd:string" />
7             <element name="wkn" nillable="true" type="xsd:string" />
8             <element name="kurs" type="xsd:double" />
9           </sequence>
10        </complexType>
11      <element name="Aktienkurs" nillable="true" type="tns1:Aktienkurs"
    />
12      </schema>
13    </types>
```

Der Inhalt eines types-Elements bietet sich immer gut zur Auslagerung an. Eine Auslagerung ist jedoch nur dann sinnvoll, wenn mehrere Web Services den gleichen Datentyp verarbeiten.

3.2.4 Das `<message>` Element

In einem `message`-Element werden die Nachrichten beschrieben, die zwischen Client und Server gesendet werden. Ein `message`-Element enthält die Parameter die in den Nachrichten übergeben werden.

Im Beispiel des RealtimeKursService werden insgesamt zwei `message`-Elemente angelegt. Es können mehrere `message`-Elemente angelegt werden, da sie sich durch den Wert ihres name-Attributs voneinander unterscheiden lassen. Das Kindelement `part` stellt die Parameter dar, die der getKurs-Methode übergeben werden.

Beispiel 17: Das message-Element

```
1     <wsdl:message name="getKursResponse">
2       <wsdl:part name="return" type="tns1:Aktienkurs" />
3     </wsdl:message>
4     <wsdl:message name="getKurstRequest">
5       <wsdl:part name="wkn" type="xsd:string" />
6     </wsdl:message>
```

3.2.5 Das `<portType>` Element

Nachdem im `message`-Element spezifiziert wird, welche Nachrichten zwischen Client und Server fließen, können mit Hilfe des `portType`-Elements die am Web Service aufrufbaren Methoden definiert werden.

Das `<operation>` Element

Eine `operation` wird als Kindelement des `portType`-Elements spezifiziert. Eine `operation` kann folgendes sein:

a. Eine Nachricht, die der Client an den Server sendet, auf der aber keine Antwort erhalten wird („One-Way“).
b. Eine Nachricht, die der Client an den Server sendet, worauf dieser ihm eine Rückantwort zusendet. Dieser Fall entspricht dem normalen Methodenaufruf mit Rückgabeparametern („Request/Response“).
c. Eine Nachricht, die der Server an den Client sendet worauf ihm der Client eine Antwort zusendet („Solicit-Response“).
d. Der Server sendet eine Nachricht an den Client („Notification“).

Alle Methoden, die der Web Service dem Client zur Verfügung stellen soll, werden mit einem `operation`-Element erfasst. Dessen Kindelemente `input` und `output` spezifizieren, welche Nachrichten (`message`-Elemente) beim Aufruf einer Methode hinein- bzw. hinausgehen.

Beispiel 18: Das portType-.und Operation- Element

```
1     <wsdl:portType name="RealtimeKursService">
2       <wsdl:operation name="getKurs" parameterOrder="wkn">
3         <wsdl:input message="intf:getKursRequest" />
4         <wsdl:output message="intf:getKursResponse" />
5       </wsdl:operation>
6     </wsdl:portType>
```

3.2.7 Das `<binding>` Element

Das binding-Element nimmt den größten Teil des WSDL-Dokuments ein. Es beschreibt wie ein Web Service (Server) mit dem Client kommuniziert. Dies schließt auch das zu verwendende Trägerprotokoll (z.B. HTTP) mit ein.

Das Element binding spezifiziert die Funktionsweise des zugrunde liegenden Web Service. Hierzu wird das style-Attribut eingefügt, welches entweder den Wert „rpc“ oder den Wert „document“ annehmen kann. Der Unterschied zwischen diesen beiden ist, dass die an den Client zurückgesendete SOAP-Nachricht anders beschrieben wird.
Das zweite Attribut transport spezifiziert das zu verwendende Trägerprotokoll.

Das `operation`-Element im `binding`-Element ähnelt dem zuvor definierten `operation`-Element (Kapitel 3.2.6). Doch es handelt sich um zwei verschiedene Elementtypen (Hervorgehoben durch Namespaces). Dieses `operation`-Element besitzt das Attribut `soapAction`, mit dem die URL der Methode des beteiligten Web Service spezifiziert werden kann.
Die Kindelemente `input` und `output` umfassen jeweils ein `body`-Element, mit dessen Attributen die Codierung der Ein- und Ausgangswerte des Web Services festgelegt wird.

Beispiel 19: Das binding-Element.

```
1   <wsdl:binding name="RealtimekursService-2SoapBinding"
    type="intf:RealtimeKursService">
2         <wsdlsoap:binding style="rpc" transport=".../http" />
3       <wsdl:operation name="getKurs">
4       <wsdlsoap:operation soapAction="" />
5       <wsdl:input>
6         <wsdlsoap:body use="encoded"
7           encodingStyle=".../encoding"
8           namespace="getKurs"
9         />
10      </wsdl:input>
11      <wsdl:output>
12        <wsdlsoap:body unse="encoded" encodingStyle=http://.../encoding
              namespace=".../services/RealtimeKursService" />
```

```
13      </wsdl:output>
14      </wsdl:operation>
15    </wsdl:binding>
```

3.2.8 Das <port> und das <service> Element

Das `service`-Element dient dazu, den Ort des Web Service zu bestimmen. Wenn der Client den Service kontaktieren möchte, muss er dessen URL als Zieladresse angeben. Mit dem `port`-Element wird der Registrierungs-`name` des Web Service angegeben, wobei das `binding`-Attribut auf das zuvor spezifizierte `binding`-Element (Kapitel 3.2.7) zeigen muss. Das `adress`-Element unterhalb von `port` enthält die URL des Web Service.

Beispiel 20: Das service- und das port-Element.

```
1     <wsdl:service name="RealtimeKursService_Service">
2       <wsdl:port name="RealtimeKursService"
3         binding="intf:RealtimeKursService-2SoapBinding">
4        <wsdlsoap:address
    location=http://localhost:8080/axis/services/RealtimeKursService />
5       </wsdl:port>
6     </wsdl:service>
```

3.3 Zusammenfassung

WSDL dient zur Beschreibung von Web Services. Im Grunde genommen ist es wieder nur ein XML-Dokument, das sich an das WSDL-Schema hält.
Ein WSDL-Dokument wird auf der Serverseite erzeugt und anschließend Clients zur Verfügung gestellt. Somit sind die Clients in der Lage nachzusehen, welche Methoden der Web Service zur Verfügung stellt und welche Parameter er in welcher Form für jede Methode erwartet.

4. UDDI

Ein Begriff der im Zusammenhang mit Web Services noch zu erwähnen ist, ist UDDI. UDDI steht für **U**niversal **D**escription, **D**iscovery and **I**ntegration. Es ist die Registry für sämtliche Web Services. In der Einleitung wurde bereits der Zusammenhang zwischen Konsument, Dienstleister und dieser Registry gezeigt.
Nachdem ein Web Service mittels WSDL definiert wurde, muss er veröffentlicht werden. Dieser Vorgang kann mit der Eintragung in eine Liste verglichen werden. Ein Beispiel aus der Nicht-Computerwelt sind die „Gelben Seiten".
WSDL hat bereits beschrieben wie die Web Services aufgebaut sind, mit denen ein Konsument den Service in Anspruch nehmen kann. UDDI ist daher ein Standard, mit dem man diese WSDL-Informationen zugänglich machen kann. Da Web Services für einen globalen Marktplatz gedacht sind, auf dem sich viele Unternehmen als Dienstleister anbieten; können in der Registry insgesamt drei verschiedene Informationstypen untergebracht werden: Business- , Service- und Technikinformationen. Diese werden auch als „White Pages", „Yellow Pages" und „Green Pages" bezeichnet.

4.1 Kernbestandteile von UDDI

Wie bereits bekannt verfügt ein Web Service Szenario immer über die drei Charaktere: Dienste-Konsument (Client), den Dienste-Anbieter (Server) und die Dienste-Registry(UDDI).

UDDI besteht aus vier Hauptspezifikationen:

- **Die Datenstruktur**: Mit ihr wird beschrieben, welche Datentypen innerhalb von UDDI gespeichert werden sollen. Ähnlich wie WSDL wird auch UDDI mit Hilfe von XML Schema beschrieben.
- **Die API-Spezifikation**: Diese Spezifikation ist eine Grundlage, wie auf eine UDDI Registry zugegriffen wird. Ein Zugriff kann auf zwei Methoden reduziert werden. Die „publishing functions" dienen dem Eintragen und Updaten des Dienstes in der Refistry. Die anderen Methoden, die „inquiry functions" führen nur Lese-Zugriffe auf UDDI-Registries durch, um die Informationen auszulesen. Die API ist programmiersprachenunabhängig und muss von API-Anbietern wie IBM in eine Zielsprache übersetzt werden.
- **Die Replikations-Spezifikation**: Diese Spezifikation ist lediglich für diejenigen gedacht, die eigene Registries aufbauen möchten. Sie umfasst alle Informationen die nötig sind, um zwischen verschiedenen UDDI-Registries Informationen duplizieren zu können.
- **Die Spezifikation für Registry-Anbieter**: Diese Spezifikation ist für diejenigen , die sich selbst als Anbieter einer UDDI Registry versuchen. Sie umfasst umfangreiche Details über einzuhaltende Sicherheitsmaßnahmen, durch die ein standardisierter Zugriff erfolgen kann.

[ECER02]

4.2 UDDI Registries

In einer UDDI Registry werden drei verschiedene Informationstypen abgelegt[wwwUDDI] :

- Die White Pages erlauben die Abspeicherung von Informationen über Unternehmen und deren Kontaktperson. Hierzu zählen auch Informationen wie Steuernummern oder die in USA typischen D.U.N.S-Nummern.
- Die Yellow Pages umfassen den eigentlichen „Service", der von dem Unternehmen angeboten wird. In der Praxis kommt es oft vor, dass ein Unternehmen nicht nur einen

Web Service anbietet, sondern gleich mehrere. Dementsprechend wird dem Konsumenten eine Möglichkeit geboten, alle Dienste dieses anbietenden Unternehmens zu erfragen.

- In den Green Pages werden die technischen Informationen des Web Service hinterlegt. Eine technische Information ist zum Beispiel ein WSDL-Dokument.

Zusammengefasst sind die "White Pages" dazu da, Web Services nach Anbietern (Unternehmen, Einzelne Personen) zu suchen. Die "Green Pages" erlauben, Web Services nach ihren Diensttypen zu suchen, und die "Green Pages" findet man Web Services nach ihren technischen Informationen aufgelistet.

4.2.1 UDDI Anbieter

IBM, Microsoft, SAP stellen eine öffentliche UDDI Registry zur Verfügung, in denen Dienste eingetragen werden können. Das Verfahren ist ähnlich dem Eintragen einer Webseite in eine Suchmaschine. Um Anwendungen für Tests zu registrieren, bieten die Registries Testaccounts an, die jedoch beschränkt sind in der Anzahl der Einträge.

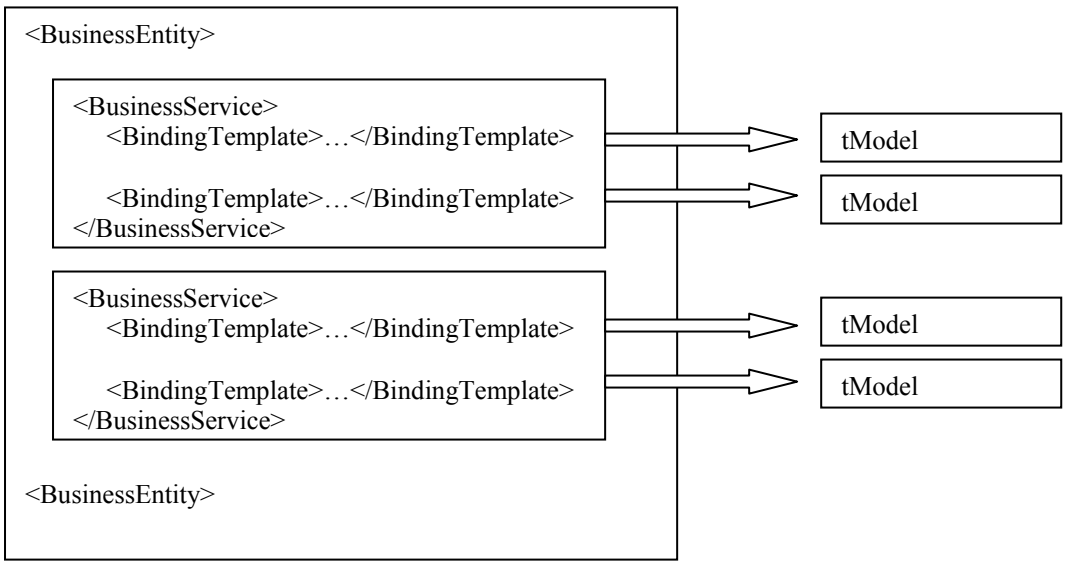
4.2.2 Eigene Registries

Eigene Registries werden entweder zum Testen oder für Intranets angewendet. Zum Beispiel wenn Web Services als Kommunikationsgrundlage eingesetzt werden, um in Großfirmen verschiedene Betriebssysteme miteinander laufen zu lassen, haben diese Firmen meist kein Interesse ihre Services in einer öffentlichen Registry anzumelden. Daher werden sich diese Unternehmen ihre eigene, private UDDI Registry aufbauen.

4.3 XML Schema einer UDDI Registry

Insgesamt gibt es vier verschiedene Elementtypen in einem UDDI-Registry-Eintrag. Ein `BusinessEntity` repräsentiert einen Eintrag in einer Registry. Es umfasst mehrere Dienste, die als `businessService` deklariert werden. Jedes `businessService`-Element verweist auf ein `tModel`-Element, in dem die technischen Spezifikationen des Web Service enthalten sind.

Abbildung 6: UDDI Daten Typen



tModel

Ein `tModel` repräsentiert eine technische Spezifikation innerhalb der UDDI Registry. Diese Spezifikation kann z.B. ein WSDL-Dokument sein, in welchem der Service technisch

beschrieben ist. Bei der Registrierung eines Dienstes wird diesem von der Registry eine eindeutige ID zugewiesen, die ihn von anderen Diensten unterscheidet.

Beispiel 21: tModel für die BeispielApplikation

```
1     <tModel
2       authorizedName=”BeispielApplikation”
3       operator=”nadaInc/services/uddi”
4       tModelKey=”UUID:123498765342829917252 4”>
5
6       <name>Ein einfaches Beispiel</name>
7
8       <overviewDoc>
9         <overviewURL>
10          http://meineFirma.com/wsdl/Echtzeitkurs.wsdl
11        </overviewURL>
12      </overvieDoc>
13
14    </tModel>
```

identifierBag und categoryBag

Um die Menge der Suchergebnisse zu reduzieren, müssen diverse Informationen von Anbieterseite in dem entsprechenden Registry-Eintrag abgelegt werden. Hierzu dienen die Elemente `identifierBag` und `categoryBag`.

Ein `identifierBag` (Bsp. 22, Zeile 1) repräsentiert eine oder mehrere identifizierende Attributmengen. Diese werden als `keyedReference`(Bsp. 22, Zeile 2) in das XML-Dokument eingetragen.

Beispiel 22: identifierBag für ein Service

```
1     <identifierBag>
2       <keyedReference>
3         <keyName>
4             <!-- Name der Abteilung, die diesen Dienst anbietet -->
5           Meine Firma Abteilung A
6         </keyName>
7         <keyValue>
8           <!-- Kurzbezeichnung des Dienstes -->
9           Stockquote_4711
10        </keyValue>
11        <tModelKey>
12          <!—RegistryKey des Dienstes -->
13          UUID:123498765342829917252 4
14        </tModelKey>
15      </keyedReference>
16    </identifierBag>
```

Es muss dabei beachtet werden, dass der Wert des Elements `tModelKey` (Bsp. X, Zeile 11)auf ein existierendes `tModel-Element` (Bsp. 21, Zeile 1)verweist.
Als weiteres kann auch ein `categoryBag` eingefügt werden. Ein `categoryBag` ist genau das Gleiche wie ein identifierBag, nur mit dem Unterschied, dass ein `categoryBag` Informationen über die Taxonomie eines Web Service enthält.

businessEntity und businessService

Ein `businessEntity` umfasst mehrere Elemente vom Typ `businessService` und klassifiziert einen einzelnen Service. Jeder `businessService` beschreibt ein Serviceangebot eine Anbieters, welches nicht unbedingt technisch sein muss.

Ein `businessService` besteht aus mehreren Kindelementen vom Typ `bindingTemplate`, in dem technische Informationen (URL des Web Service, Referenzen auf `tModel`-Elemente usw.) untergebracht werden.

4.4 Zusammenfassung

UDDI stellt potentiellen Clients die erzeugten WSDL-Dokumente zur Verfügung. Clients sind dann in der Lage nachzusehen, welche Methoden der Web Service zur Verfügung stellt und was dieser vom Client erwartet.
Besonders begünstigt werden die B2B Applikationen, da damit eigentlich die Interaktion mit Menschen für diese Dienste entfällt. UDDI stellt eine automatisierte Kommunikation zwischen Computern dar.
Trotzdem müsste sich diese Art der Kommunikation im Gegensatz zur herkömmlichen Methode: “suchen, finden, anrufen, Termin ausmachen, Preis aushandeln“, erstmals durchsetzen.

5. Literaturangabe

SOAP

[S002] SOAP Version 1.2 Part 0: Primer; Candidate Recommendations, XML Protocol Working Group (2002/12/19).
[S102] SOAP Version 1.2 Part 1: Messaging Framework; Candidate Recommendations, XML Protocol Working Group (2002/12/19).
[S202] SOAP Version 1.2 Part 2: Adjuncts; Candidate Recommendations, XML Protocol Working Group (2002/12/19).

WSDL

[WSDL03] Web Services Description Language Version 1.2; Working Draft, Web Services Description Working Group (2003/03/03).
[WSDLB03]Web Services Description Language Version 1.2: Bindings; Working Draft, Web Services Description Working Group (2003/01/24).
[WSDR02] Web Services Description Requirements; Working Draft, Web Services Description Working Group (2002/10/28).

UDDI

[wwwUDDI] www.uddi.org

Others

[SG02] Buidling Web Services with Java, Steve Graham et al., Sams Publishing, 2002.
[ECER02] Web Services Essentials, Ethan Cerami, O'reilly, February 2002.
[EN02] Understanding Web Services: XML, WSDL, SOAP, and UDDI, Eric Newcomer, May 2002.
http://ws.apache.org/axis
www.bitpipe.com, Web Services White Papers

Anhang A: Datentypen in XML Schema

Die Datentypen die in XML Schema verwendet werden können, sind in folgender Tabelle aufgelistet.

Simple Type	Beispiel	Anmerkung
string	Dies ist ein String	
normalizedString	Dies ist ein String	
token	Dies ist ein String	
byte	-1, 126	
unsignedByte	0, 126	
Base64Binary	GpM7dewysADF	
positiveInteger	1, 126789	
negativeInteger	-213456, -1	
nonNegativeInteger	0, 1, 123232	
nonPositiveInteger	-123232, -1, 0	
int	-1, 123123132323	
short	-1, 12345	
unsignedShort	0, 1234	
decimal	-1.23, 0, 123.5, 1000.0	
float	-INF, -1E4, -0, 0 12.78E-2, 12, INF, NaN	Äquivalent zu einer 32Bit langen Fließkommazahl
double	-INF, -1E4, -0, 0 12.78E-2, 12, INF, NaN	Äquivalent zu einer 64Bit langen Fließkommazahl
bollean	True, false, 1, 0	
time	12:20:00.123, 12:20:00.123-05:00	
dateTime	2003-04-15T13:20:00.000-05:00	15. April 2003 13:20 5 Stunden hinter del Weltzeit
duration	P1Y2M3DT10H30M12.3S	1 Jahr, 2 Monate, 3 Tage, 10 Stunden, 30 Minuten, 12.3 Sekunden
date	2003-04-15	
gMonth	--04--	April
gYear	2003	
gYearMonth	2003-04	April 2003
gDay	---15	
gMonthDay	--04-15	
Name	abcIchBinEinName	XML 1.0 Name Type
QName	DeutscheBank:Konto	XML 1.0 Namespace QName
NCName	Konto	XML Namespace NCName
anyURI	http://www.xyz.de	
language	en-US, de, fr	Sprachkennungzeichen laut xml:lang 1.0
ID		
IDREF		
IDREFS		
ENTITY		
ENTITIES		
NOTATION		
NMTOKEN	US, Detuschland	
NMTOKENS	US AirForce, Deutschland Spanien Frankreich	