10. Web Services

Markus Keidl, Alfons Kemper, Stefan Seltzsam, Konrad Stocker
Universität Passau

10.1. Einleitung

In den letzten Jahren hat sich das Internet immer mehr zu einer Plattform entwickelt, auf der Dienste angeboten werden. Bisher verwenden Dienste im Internet meist eine Kombination aus HTML-Seiten und HTML-Formularen als Schnittstelle, da sie für die Anzeige in einem Browser und die Interaktion mit einem menschlichen Benutzer konzipiert wurden. Aus Gründen der Effizienzsteigerung wollen aber viele Firmen mittlerweile Dienste automatisiert nutzen, also ohne menschliche Interaktion, und eigene Dienste schnell und unkompliziert über das Internet zur Verfügung stellen. Für diesen Zweck sind Formulare ungeeignet, da für jeden Dienst eigene, spezifische Formulare entwickelt werden müssen. Dies erschwert zum einen das Bereitstellen von Diensten und zum anderen deren automatisierte Nutzung und die Ermittlung von Ergebnissen oder Fehlermeldungen aus den angezeigten HTML-Seiten. Zur Ermittlung der interessanten Informationen einer HTML-Seite müssen aufwändige „screen scraping"-Techniken eingesetzt werden. Diese sind gegenüber Änderungen des Designs der HTML-Seiten nicht robust und müssen so immer wieder angepasst werden.

Um vollautomatische Dienstnutzung und Dienstkomposition (Interoperabilität) zu ermöglichen, wird derzeit immer öfter eine neue Technologie eingesetzt: Web Services (im Folgenden auch Web-Dienste oder Dienste genannt). Viele große Softwarefirmen haben mittlerweile entsprechende Produkte im Angebot oder entwickeln gerade ihre eigene Web-Service-Lösung. Als die bekanntesten Vertreter sind hier sicherlich BEA WebLogic, HP Web Services Platform, IBM WebSphere, Microsoft .NET, mySAP.com von SAP und SUN One zu nennen. Bisher gibt es keine einheitliche Definition des Begriffs Web Service. Im üblichen Sprachgebrauch bezeichnet ein Web Service allerdings einen Dienst, der Benutzern über das Web zur Verfügung gestellt wird und dabei beispielsweise auf XML [KeEi01] und HTTP zurückgreift. Web Services unterscheiden sich dabei von klassischen Diensten im Web dadurch, dass sie nicht auf die Benutzung durch Menschen, sondern auf eine automatisierte Benutzung ausgerichtet sind. Ein weiteres Ziel von

Web Services ist die Interoperabilität, das heißt, Web Services sollen unabhängig vom Betriebssystem, der Programmiersprache, in der die Services entwickelt worden sind, und der Web Service Engine – so bezeichnet man eine Anwendung, die Web Services in einem Netzwerk verfügbar macht – in einer standardisierten Weise genutzt werden können und auch miteinander interagieren können. Ein bekanntes Beispiel für Web Services sind Marktplätze, z. B. Marktplätze der Automobilindustrie, die dazu dienen, den Einkauf teilnehmender Automobilfirmen zu optimieren und automatisieren, d. h. möglichst schnell das günstigste Produktangebot aus den Angeboten aller Zulieferer zu finden. Marktplätze automatisieren auch den Bestellvorgang und die Abrechnung, was zu weiteren Einsparungen und schnellerer Abarbeitung führt. Um dies zu leisten, integrieren Marktplätze die Daten und ERP-Systeme (Enterprise-Resource-Planning-Systeme) der beteiligten Anbieter in ein zentrales System. [KeWi01, WiWK02] beschreiben eine alternative Architektur für Marktplätze, die keine vollständige Datenintegration am Marktplatz erfordert. Sie erlaubt Anbietern Teile der Daten in ihren lokalen Systemen zu belassen und ermöglicht es dadurch beispielsweise, Preise sehr dynamisch zu kalkulieren. Andere bedeutende Anwendungsbereiche von Web Services sind unter anderem Anwendungsintegration, elektronischer Datenaustausch, Electronic-Business-Anwendungen und Business-to-Business-Integration. Für derartige Anwendungen sind Qualitätskriterien und -garantien, z. B. Antwortzeitgarantien [KSWD02], ein wichtiger Aspekt, den wir hier allerdings nicht betrachten werden, da er über den Rahmen dieses Kapitels hinausgeht.

Um die Interoperabilität von Diensten zu gewährleisten, sind Standards nötig. Mittlerweile existieren mehrere verschiedene, auf XML basierende Standards: SOAP (Simple Object Access Protocol von IBM, Microsoft, …), ebXML (Electronic Business using eXtensible Markup Language von OASIS und UN/CEFACT), UDDI (Universal Description, Discovery and Integration von HP, IBM, Intel, Microsoft, SAP, Software AG, Sun, …), WSDL (Web Services Description Language von Ariba, IBM und Microsoft), WSFL (Web Services Flow Language von IBM), XLANG (Microsoft), WS-Inspection (Web Service Inspection Language von IBM und Microsoft) und einige andere. Die Bedeutung der Interoperabilität zeigt sich auch in Firmenkooperationen, die es sich zur Aufgabe machen Referenzarchitekturen basierend auf den bestehenden Standards zu entwickeln. Ein Beispiel hierfür ist WS-I[1] (Web Services

[1] Siehe `www.ws-i.org`

Interoperability Organization), zu deren Mitgliedern unter anderem Accenture, DaimlerChrysler und SAP zählen.

Wir werden in diesem Kapitel Web-Dienste exemplarisch auf der Basis von SOAP, UDDI, WS-Inspection und WSDL erläutern, da diese Standards schon sehr weit verbreitet sind und auch die meisten der oben genannten Web-Service-Lösungen darauf basieren oder diese Protokolle zumindest unterstützen. Grundlage für die Interoperabilität ist ein einheitliches Kommunikationsprotokoll (in unserem Fall SOAP), das es ermöglicht, auf eine standardisierte Weise mit Diensten zu kommunizieren. Es ist natürlich nicht ausreichend, Dienste nur zur Verfügung zu stellen, man muss es potentiellen Nutzern auch ermöglichen diese Dienste zu finden. Bisher wurden Informationen im Internet durch Suchmaschinen oder Web-Kataloge indexiert und damit auffindbar gemacht. In der Welt der Web Services übernehmen diese Aufgaben beispielsweise WS-Inspection und UDDI. Hat man einen Dienst gefunden, benötigt man noch die Information, welche Eingabedaten der Dienst erwartet und wie er sein Ergebnis zur Verfügung stellt. Diese Dienstbeschreibung kann beispielsweise mit Hilfe von WSDL erfolgen.

10.2. Beispiel-Szenario

Um die Protokolle und Abläufe beim Anbieten und Nutzen von Web Services nicht nur abstrakt darstellen zu können, verwenden wir einen Temperaturdienst als Beispiel-Szenario und zeigen anhand dieses Dienstes, was ein Anbieter bzw. ein Benutzer eines Web Services tun muss und wie die verschiedenen Protokolle verwendet werden. Informationen zu den in diesem Kapitel vorgestellten Diensten (inklusive aller notwendigen XML-Dokumente) bieten wir auch über unseren Server `wetter.fmi.uni-passau.de` an, auf dem diese Dienste auch betrieben werden.

Der Temperaturdienst liefert zu einem vom Benutzer vorgegebenen Zeitpunkt die Temperatur zurück, die an einem bestimmten Temperatursensor in der Stadt Passau gemessen wurde. Hat der Dienst für den geforderten Zeitpunkt keine Messdaten zur Verfügung, liefert er die Temperatur zum nächstgelegenen Zeitpunkt zurück, zu dem ein Messwert existiert. Weitere Dienste in unserem Beispiel-Szenario sind ein Einheitenumrechnerdienst und ein TemperaturInFahrenheitDienst. Der Einheitenumrechnerdienst rechnet Temperaturangaben in verschiedene Einheiten um, der TemperaturInFahrenheitDienst liefert

wie der Temperaturdienst die Temperatur in der Stadt Passau zu einem gegebenen Zeitpunkt, allerdings in Grad Fahrenheit.

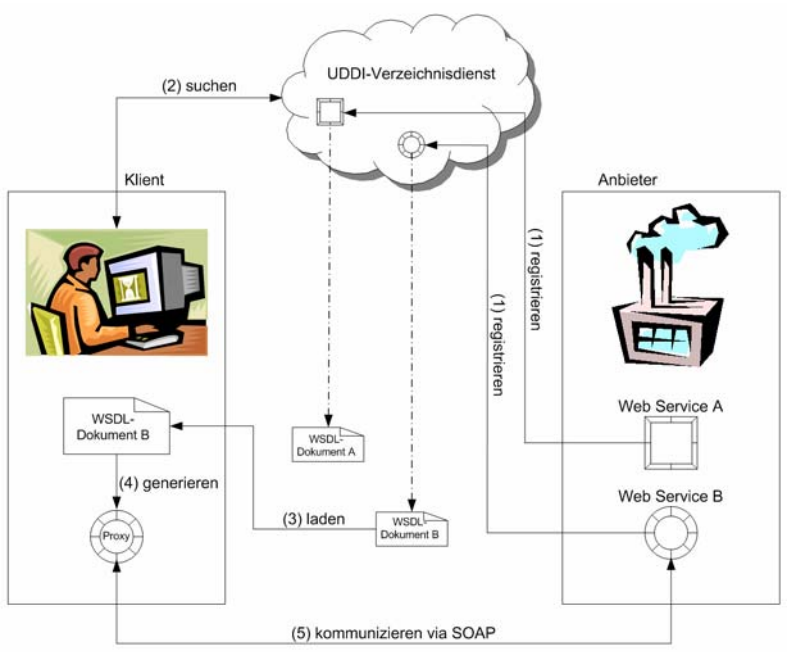


Abbildung 10.1: Überblick über den Einsatz von Web Services

Abbildung 10.1 gibt einen Überblick über alle nötigen Aktionen, um einen Web Service zur Verfügung zu stellen und diesen zu nutzen. Details zu diesen Schritten findet man in den jeweiligen Abschnitten dieses Kapitels. Ein Dienstanbieter muss seine Dienste (im Bild Web Service A und Web Service B) bei einem UDDI-Verzeichnisdienst registrieren, um die Informationen verfügbar zu machen, welche Dienste er anbietet und wie man diese ansprechen kann. Wie die Kommunikation mit den Diensten aussehen muss, ist dabei in WSDL-Dokumenten beschrieben, die nicht im UDDI-Verzeichnis selbst, sondern „irgendwo“ im Internet gespeichert sind. Will nun ein Klient einen Dienst nutzen, sucht er sich einen für seine Zwecke geeigneten Dienst aus dem UDDI-Verzeichnis aus. Nach der Auswahl eines Dienstes (in der Abbildung Web Service B), wird das zugehörige WSDL-Dokument geladen und dazu verwendet, einen Proxy für den Web Service zu generieren. Diese beiden Schritte können bereits automatisiert werden. Der generierte Proxy wird verwendet, um mit dem tatsächlichen Web Service via SOAP-Nachrichten zu kommunizieren. Die gerade aufgezählten Schritte werden nun – zum einen aus der Sicht des Dienstanbieters, zum anderen aus Klientensicht – detaillierter beschrieben.

Abbildung 10.2 zeigt eine Übersicht über die Aktionen, die ein Anbieter ausführen muss, damit er einen Dienst anbieten kann. Zuerst benutzt der Anbieter einen UDDI-Verzeichnisdienst, um zu überprüfen, ob es bereits ein so genanntes tModel (siehe Abschnitt 10.4.1) gibt, das die Art von Dienst beschreibt, die er anbieten will. Sollte dies der Fall sein, verwendet er dieses tModel inklusive des zugeordneten WSDL-Dokumentes als Grundlage für seinen Dienst. Sollte kein geeignetes tModel existieren, muss der Dienstanbieter ein neues tModel und ein dazugehörendes WSDL-Dokument erzeugen und bei dem UDDI-Verzeichnisdienst registrieren. Auf der Basis des WSDL-Dokumentes kann sich der Dienstanbieter durch ein geeignetes Werkzeug ein Gerüst des Web Services generieren lassen, also beispielsweise eine Java-Klasse. Diese Klasse genügt dann bereits der Schnittstelle, die im WSDL-Dokument festgelegt ist. Nun muss der Dienstbetreiber das Gerüst noch vervollständigen, also die Funktionalität des Dienstes implementieren. Danach kann er den Web Service betreiben. Damit der Dienst auch von anderen gefunden werden kann, muss er anschließend noch bei einem UDDI-Verzeichnisdienst registriert werden.

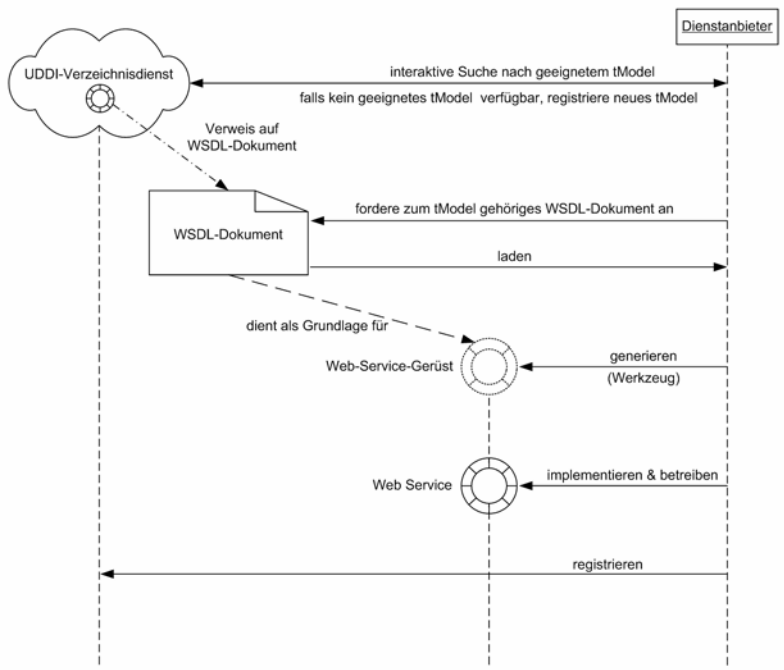


Abbildung 10.2: Aktionen des Dienstanbieters

Der gerade skizzierte Weg beschreibt nur eine Möglichkeit, einen Dienst im Netz verfügbar zu machen. Wenn beispielsweise eine Anwendung

bereits existiert und als Dienst verfügbar gemacht werden soll, kann auch ein WSDL-Dokument aus dem vorhandenen Code generiert werden. Für dieses WSDL-Dokument kann man dann ein entsprechendes tModel beim UDDI-Verzeichnisdienst registrieren.

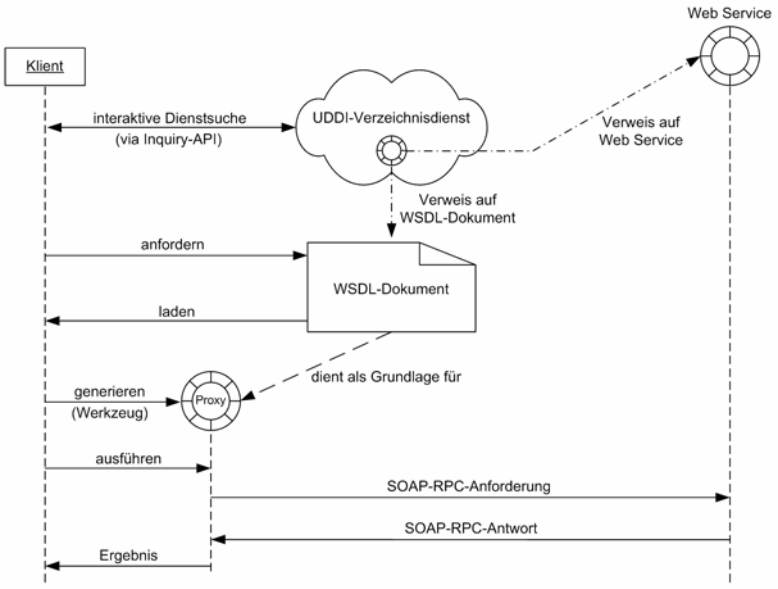


Abbildung 10.3: Aktionen von Dienstbenutzern

Abbildung 10.3 zeigt, ähnlich einem Sequenzdiagramm, die Aktionen, die Klienten ausführen müssen, um einen Dienst nutzen zu können. Wie bereits erwähnt suchen sie dafür nach einem geeigneten Dienst in einem Dienstverzeichnis. Dazu sprechen sie dieses Verzeichnis mit Hilfe der so genannten Inquiry-API an. Wurde ein Dienst gefunden, kann das zugehörige WSDL-Dokument aus dem Internet geladen werden. Wo dieses Dokument zu finden ist, ist im UDDI-Verzeichnis abgelegt. Das WSDL-Dokument spezifiziert, wie Nachrichten an den ausgewählten Dienst aussehen müssen. Im UDDI-Verzeichnis steht, unter welcher URL der Dienst im Internet erreichbar ist. Unter Verwendung eines geeigneten Werkzeugs kann man aus den Informationen des WSDL-Dokumentes und der URL aus dem UDDI-Verzeichnis einen Proxy für die Interaktion mit dem Web-Dienst generieren lassen. Dies kann zum Beispiel eine Java-Klasse mit einer Methode sein, die alle für den Aufruf des Dienstes erforderlichen Parameter übergeben bekommt. Der Proxy kümmert sich dann darum, dass diese Parameter im richtigen Format in eine SOAP-RPC-Nachricht (siehe 10.3.2) verpackt an den Web Service geschickt werden. Außerdem verarbeitet der Proxy das Ergebnis des Aufrufes und wandelt es beispielsweise in ein Java-Objekt um. Auf diese

Weise verbirgt der Proxy, dass überhaupt ein Web Service benutzt wird.

Im Folgenden werden die verschiedenen erwähnten Standards anhand des Temperaturdienst-Beispiels detaillierter beschrieben. Der nächste Abschnitt erklärt das Kommunikationsprotokoll SOAP, das von Diensten zum Datenaustausch verwendet wird. Anschließend werden in Abschnitt 10.4 zwei Möglichkeiten zur Dienstverwaltung beschrieben: UDDI als zentrales Dienstverzeichnis und WS-Inspection als Möglichkeit, auf einem Web-Server in definierter Weise auf lokale Dienste zu verweisen. Abschnitt 10.5 erläutert wie Dienste mit Hilfe von WSDL beschrieben werden können. Der darauf folgende Abschnitt beschreibt die Dienstkomposition und –interaktion, d. h. wie neue, zusammengesetzte Dienste basierend auf existierenden Diensten entwickelt werden können. In Abschnitt 10.7 wird kurz auf verschiedene Plattformen, Produkte und Infrastrukturen für Web Services eingegangen und als Beispiel für eine Dienstplattform der Forschungsprototyp ServiceGlobe vorgestellt. Abschnitt 10.8 beendet dieses Kapitel mit einer Zusammenfassung und einem Ausblick.

10.3. Datenaustausch (SOAP)

SOAP (Simple Object Access Protocol) ist ein Kommunikationsprotokoll für verteilte Anwendungen, das es ermöglicht, strukturierte und typisierte Daten mit Hilfe von XML auszutauschen. Der große Vorteil von SOAP besteht darin, dass verschiedene Protokolle wie z. B. HTTP, SMTP (Simple Mail Transfer Protocol) und FTP (File Transfer Protocol) als Übertragungsprotokoll verwendet werden können. Damit ist es mit SOAP sogar möglich, durch die meisten Firewalls hindurch Nachrichten zu verschicken, was bei anderen Protokollen im Bereich verteilter Anwendungen wie z. B. IIOP (Internet Inter-ORB Protocol) normalerweise nicht möglich ist.[2]

Für den Datenaustausch definiert das SOAP-Protokoll, wie Objekte serialisiert werden, also wie Objekte bzw. vernetzte Objektstrukturen (sequentiell) auf XML abgebildet werden und umgekehrt. SOAP definiert selbst keine Anwendungssemantik und kann dadurch für ein breites Anwendungsspektrum vom einfachen Zustellen einer Nachricht

[2] Allerdings entstehen dadurch auch Sicherheitsrisiken, die durch die bisher verfügbaren Firewall-Lösungen nicht hinreichend abgedeckt werden.

bis zum entfernten Prozeduraufruf (Remote Procedure Call; RPC) verwendet werden. Wir beschränken uns aus Platzgründen auf die Beschreibung der grundlegenden Funktionalität von SOAP – weitergehende Informationen findet man beispielsweise in der SOAP-Spezifikation [BEKL00] oder in dem Buch [ScSt00].

10.3.1. Nachrichtenformat

Der Aufbau einer SOAP-Nachricht sieht folgendermaßen aus:

```
<soap:Envelope  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!--Der Header ist optional -->
  </soap:Header>

  <soap:Body>
    <!-- Serialisierte Objektdaten -->
  </soap:Body>
</soap:Envelope>
```

SOAP-Dokument 10.1: Aufbau einer SOAP-Nachricht

Das Envelope-Element ist das Wurzelelement der Nachricht und kann neben einem Body-Element ein optionales Header-Element enthalten. Durch das encodingStyle-Attribut kann angegeben werden, wie Objekte zu XML serialisiert worden sind, um in dieser Nachricht verschickt zu werden. Dieses Attribut kann auch innerhalb anderer Elemente verwendet werden und gilt dann nur für den entsprechenden Teil der Nachricht. Der im Beispiel angegebene encodingStyle entspricht der Standard-Serialisierung von SOAP (also der Serialisierung, die in der Spezifikation angegeben ist), er muss aber trotzdem angegeben werden. Außerdem wird in dem Envelope-Element noch das Kürzel soap für den SOAP-Namensraum definiert.

Der Header einer SOAP-Nachricht bietet einen generischen Mechanismus, um SOAP dezentral erweitern zu können, ohne diese Erweiterungen vorher mit anderen Kommunikationspartnern abstimmen zu müssen. SOAP definiert einige Attribute, die es erlauben anzugeben, wie der Empfänger mit Header-Erweiterungen umgehen soll und ob diese Erweiterungen verpflichtend verstanden werden müssen. Beispiele für derartige Erweiterungen sind z. B. Authentifizierung, Abrechnung oder Transaktionsmanagement. Ein weiteres Beispiel ist die Web Service Security Language [ABDK02], die SOAP-Nachrichten mit sicherheitsrelevanten Informationen anreichert. Die Möglichkeiten des Header-Elementes gehen allerdings über den Rahmen dieser SOAP-Einführung hinaus. Die interessierten Leser werden auf weiterführende Literatur verwiesen [BEKL00, ScSt00]. Die SOAP-Nachrichten in

diesem Abschnitt beinhalten der Einfachheit halber kein Header-Element.

Das Body-Element einer SOAP-Nachricht bietet eine einfache Möglichkeit, Daten zwischen dem Sender und dem Empfänger auszutauschen. Typische Verwendungszwecke des Body-Elementes sind die Aufnahme von serialisierten Daten für RPC-Aufrufe oder von Fehlerbenachrichtigungen. SOAP selbst definiert nur ein Element, das innerhalb des Body-Elementes vorkommen kann: das Fault-Element. Dieses Element wird zur Übermittlung von Fehlerzuständen genutzt und beinhaltet weitere Unterelemente, deren Beschreibung über den Rahmen dieser Einführung hinausgeht.

Die Standard-Serialisierung von Daten basiert auf einem einfachen Typsystem, das eine Generalisierung der gebräuchlichen Typsysteme von Programmiersprachen, Datenbanken und semistrukturierten Datenmodellen darstellt. Ein Typ ist dabei entweder ein einfacher (skalarer) Typ, z. B. String oder Integer, oder ein zusammengesetzter Typ, z. B. Adresse. SOAP legt unter anderem fest, wie Arrays und Referenzen abgebildet werden und wie komplette Graphen von Datenobjekten dieses Typsystems in XML abgebildet werden und umgekehrt. Die Abbildung vom Typsystem einer Programmiersprache in das Typsystem, das der Standard-Serialisierung zugrunde liegt, ist nicht spezifiziert und muss für Typen, die über die Typen der Standard-Serialisierung hinausgehen, festgelegt werden. Eine detaillierte Beschreibung der SOAP-Standard-Serialisierung findet man in der SOAP-Spezifikation [BEKL00].

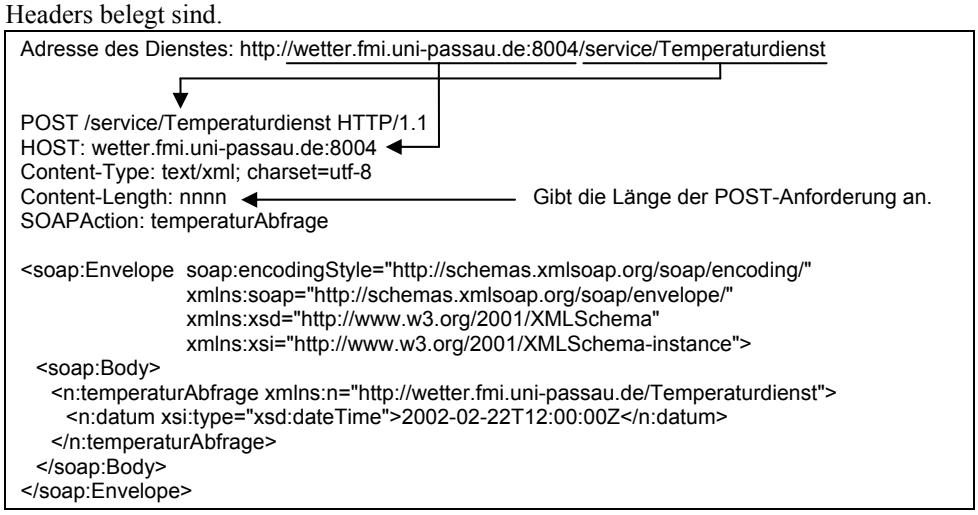
10.3.2. Kommunikation mit SOAP

Der SOAP-Standard legt nicht fest, welches Protokoll zum Versenden von Nachrichten verwendet werden soll, gibt aber als eine standardisierte Möglichkeit die Einbettung von SOAP in HTTP an. Dabei werden HTTP-POST-Anforderungen[3] für den Transport von SOAP-Nachrichten an einen Server verwendet und HTTP-Antworten liefern das Ergebnis an den Klienten zurück. SOAP-Nachrichten könnten allerdings auch mit Hilfe eines Mailprotokolls (SMTP) oder eines Dateiübertragungsprotokolls (FTP) transportiert werden. Diese

[3] Eine HTTP-POST-Anforderung ist eine spezielle Nachricht des HTTP-Protokolls, die zur Übertragung von größeren Datenmengen genutzt wird.

Möglichkeiten sind bisher allerdings nicht standardisiert.

Die HTTP-Einbettung von SOAP nutzt die vielfältigen Möglichkeiten von HTTP, ohne dabei dessen Semantik zu ändern. Der bestehende Standard wird lediglich dazu benutzt SOAP-Nachrichten als Nutzlast zu transportieren. Außerdem wird durch das SOAP-Protokoll festgelegt, wie SOAP (zusammen mit der HTTP-Einbettung) für RPC-Aufrufe genutzt werden kann. Dabei werden die URI des Ziel-Objektes und die Methode, die aufgerufen werden soll, spezifiziert und die Parameter für die Methode übergeben. Nachfolgend zeigen wir zwei Nachrichten eines SOAP-RPC-Aufrufs unseres Temperaturdienstes. Die SOAP-Dokumente sind in eine HTTP-POST-Anforderung bzw. eine HTTP-Antwort eingebettet. SOAP-Dokument 10.2 zeigt die Nachricht, die an den Temperaturdienst geschickt wird und gibt an, wie die Felder des HTTP-Headers belegt sind.



SOAP-Dokument 10.2: SOAP-Nachricht, eingebettet in eine HTTP-POST-Anforderung

Das Feld SOAPAction gibt die Bestimmung der SOAP-Nachricht an. Hier kann eine beliebige URI stehen, in unserem Fall ist es der Name der Methode, die aufgerufen werden soll. Das Envelope-Element spezifiziert die Namensräume und die verwendete Serialisierung für das Dokument. Das Body-Element enthält ein Unterelement, das genauso heißen muss wie die Methode, die aufgerufen werden soll, also temperaturAbfrage. Innerhalb dieses Elementes werden alle Parameter in derselben Reihenfolge und mit demselben Namen aufgezählt, wie in der Methodendeklaration angegeben. Bei unserem Temperaturdienst-Beispiel ist dies nur ein Parameter mit dem Namen datum (vom Typ dateTime). Insgesamt ist diese SOAP-Nachricht also eine Anfrage an den

Temperaturdienst, die am 22.02.2002 um 12.00 Uhr gemessene Temperatur zu liefern.[4]

Die Antwort des Dienstes ist in dem SOAP-Dokument 10.3 dargestellt: Wie der HTTP-Header anzeigt (HTTP/1.1 200 OK), wurde die Anforderung erfolgreich bearbeitet. Die Antwort des Dienstes findet man innerhalb des Body-Elementes. Der Name des „Antwortelementes“ wird üblicherweise aus dem Namen des „Abfrageelementes“ und einem angehängten „Response“ gebildet, in unserem Fall also temperaturAbfrageResponse. Das Ergebnis der Abfrage wird üblicherweise innerhalb eines Elementes mit dem Namen Result zurückgegeben. Die beiden Elementnamen sind allerdings zur Auswertung des Ergebnisses nicht relevant. Wie in der Antwort zu sehen ist, wurden am 22.02.2002 um 13.00 Uhr (also dem nächstgelegenen verfügbaren Messtermin zum 22.02.2002, 12.00 Uhr) 4.4 Grad Celsius gemessen.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: nnnn

<soap:Envelope  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <n:temperaturAbfrageResponse xmlns:n="http://wetter.fmi.uni-passau.de/Temperaturdienst">
      <Result   xsi:type="ns2:TemperaturdienstAntwort"
                xmlns:ns2="http://wetter.fmi.uni-passau.de/Temperaturdienst.xsd">
        <temperatur xsi:type="xsd:double">4.4</temperatur>
        <einheit xsi:type="xsd:string">Celsius</einheit>
        <messdatum xsi:type="xsd:dateTime">2002-02-22T13:00:00Z</messdatum>
      </Result>
    </n:temperaturAbfrageResponse>
  </soap:Body>
</soap:Envelope>
```

SOAP-Dokument 10.3: SOAP-Nachricht, eingebettet in eine HTTP-Antwort

10.4. Dienstverwaltung

Um die Nutzung von Diensten zu ermöglichen, müssen Informationen über angebotene Dienste gefunden werden können. Diesen Zweck erfüllen Verzeichnisdienste für Dienstinformationen. Die UDDI-

[4] Das "Z" am Ende der Datumsangabe in der SOAP-Nachricht zeigt an, dass Coordinated Universal Time (UTC) (früher auch Greenwich Mean Time (GMT) genannt) verwendet wird.

Initiative (Universal Description, Discovery and Integration) [UDDI] hat zum Ziel, ein globales Verzeichnis für solche Dienst-Metadaten zu etablieren. Dieser Initiative haben sich bereits mehr als 300 Firmen angeschlossen. Darunter befinden sich Branchengrößen wie HP, IBM, Microsoft, SAP und Software AG.

Eine Aufgabe des UDDI-Verzeichnisdienstes ist die Speicherung von Dienst-Metadaten in einer einheitlichen Datenstruktur (UDDI-Schema) an zentralen und öffentlich zugänglichen Stellen im Internet (UDDI-Server) durch einheitliche Veröffentlichungs-Mechanismen (UDDI-Publishing-API). Eine weitere Aufgabe ist die Unterstützung der Metadatenabfrage durch eine normierte Anfragesprache (UDDI-Inquiry-API).

Dienstanbieter können Metadaten ihrer Dienste auch – alternativ oder zusätzlich zu der Speicherung in einem globalen UDDI-Verzeichnis – in standardisierter Form auf ihrem Web-Server anbieten, indem sie dort entsprechende Dateien (WS-Inspection-Dokumente) ablegen. Bei Inspektion des Web-Servers, beispielsweise durch eine Suchmaschine, können diese Dateien, die Verweise auf Informationen über die verfügbaren Dienste eines Anbieters enthalten, ausgewertet und einem Benutzer verfügbar gemacht werden (siehe Abschnitt 10.4.2).

Während also ein UDDI-Verzeichnisdienst einen globalen „Wer-liefert-welchen-Web-Service“ Katalog darstellt, bietet WS-Inspection eine strukturierte Methode, um auf Informationen über Dienste eines Anbieters zu verweisen.

10.4.1. Dienstverzeichnis UDDI

Das Ziel der UDDI-Initiative ist die Festlegung eines Standards für Verzeichnisdienste von Web Services. Aus konzeptueller Sicht definiert UDDI ein verteiltes Datenbanksystem zur Speicherung von Dienst-Metadaten basierend auf offenen Standards und Protokollen. Wesentliche Eigenschaften eines solchen Systems wie ein globales und einheitliches Datenschema, eine Anfragesprache, ein Autorisierungskonzept und eine Replikationsstrategie werden dazu in UDDI definiert. Ein Transaktionskonzept lässt UDDI jedoch vermissen. Da Änderungen von Daten nur relativ selten und von wenigen Berechtigten auf voneinander unabhängigen Datenbeständen durchgeführt werden und die breite Öffentlichkeit nur Leseberechtigung besitzt, fällt dies in der Praxis nicht ins Gewicht. Um eine globale Verfügbarkeit des Verzeichnisses zu gewährleisten, ist geplant, viele lokale Installationen von UDDI-Servern zu einem globalen Verbund zusammenzuschließen, dessen Daten

weltweit periodisch abgeglichen werden. Dieser Verbund, UDDI-Wolke genannt, soll dem Anwender wie ein einzelner UDDI-Server erscheinen. Abbildung 10.4 zeigt diesen Verbund schematisch. Natürlich ist es auch möglich, eine lokale Installation unabhängig vom globalen Verbund zu betreiben, beispielsweise um Informationen über Dienste nur innerhalb eines Intranets zur Verfügung zu stellen.

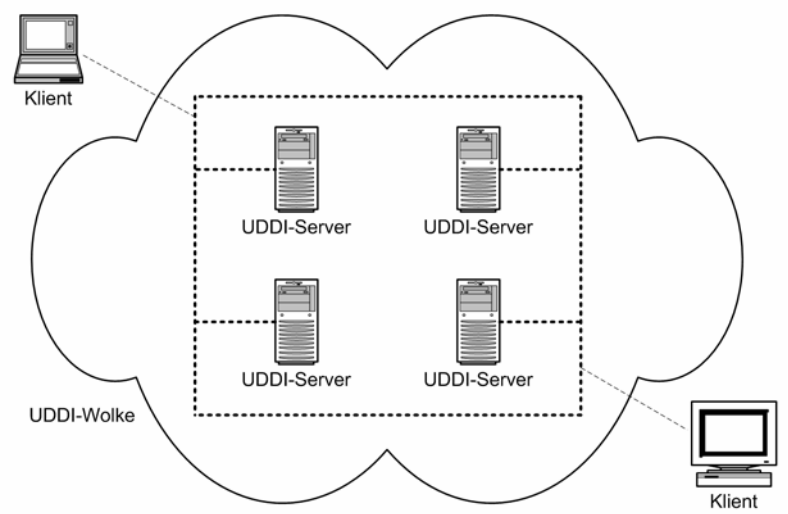


Abbildung 10.4: UDDI-Server-Verbund

In den folgenden Abschnitten werden die wesentlichen Komponenten des „Datenbanksystems“ UDDI vorgestellt. Nach einer Beschreibung des Datenschemas wird kurz die Anfragesprache vorgestellt und abschließend ein Überblick über das Replikationsverfahren gegeben.

Daten in UDDI

Eine wesentliche Voraussetzung für einen globalen Verzeichnisdienst ist die Spezifikation eines einheitlichen Schemas der zu speichernden (Meta-)Daten. Ein UDDI-Verzeichnisdienst unterscheidet konzeptionell drei verschiedene Klassen von Informationen:

- White Pages: Diese Klasse umfasst Daten über den Dienstanbieter (d. h. in der Regel eine Firma). Sie enthält neben Adressdaten und Daten über Kontaktpersonen eventuell auch weitere Identifikatoren von Unternehmen wie etwa Steuernummern oder ähnliches.
- Yellow Pages: Diese Klasse von Daten entspricht dem gedruckten Pendant – den gelben Seiten – insoweit, als sie verschiedene

industrielle Kategorisierungen basierend auf Standard-Taxonomien umfasst (z. B. Universal Standard Products and Services Classification; UNSPSC). Im Gegensatz zur gedruckten Variante sind hier jedoch beliebig feine und vom Anbieter selbstdefinierte Kategorisierungen möglich.

– Green Pages: Zusätzlich zu den mehr administrativen Informationen der White Pages und Yellow Pages werden in einem UDDI-Verzeichnis auch technische Informationen über die angebotenen Dienste abgelegt. Dabei können sowohl Spezifikationsdokumente referenziert als auch Daten über konkrete Verfügbarkeitsorte, die so genannten Zugriffspunkte eines Dienstes, abgelegt werden. Details über die Spezifikationsdokumente, welche in der Regel WSDL-Dokumente sind, finden sich in Abschnitt 10.5.

Jede dieser Informationsklassen wird in einem UDDI-Verzeichnis auf festgelegte Datenstrukturen abgebildet. Ein wesentliches Prinzip des UDDI-Ansatzes ist die Verwendung von offenen und verbreiteten Standard-Internet-Protokollen. Deshalb werden alle Daten im XML-Format ausgetauscht und abgelegt. Als Kommunikationsprotokoll zwischen UDDI-Server und Klient kommt SOAP zum Einsatz.

Datenstrukturen

Wie bereits erwähnt werden alle erfassten Daten auf das UDDI-XML-Schema abgebildet. Dieses umfasst fünf zentrale Datenstrukturen[5], deren Instanzen global eindeutig durch Universally Unique IDs (UUIDs) identifiziert werden.[6] Diese UUIDs entsprechen Identifikatoren in Datenbanksystemen. UDDI definiert die Strukturen businessEntity für Fimeninformationen, businessService für Klassen von angebotenen Diensten, bindingTemplate für technische Informationen, tModel (Abkürzung für „technical model“) zur Kategorienbildung und Referenzierung von technischen Informationen und publisherAssertion für die Modellierung von Geschäftsbeziehungen zwischen verschiedenen Firmen. Diese fünf Datenstrukturen zusammen mit kleineren Hilfsstrukturen sind in Abbildung 10.5 als UML-Klassendiagramm dargestellt.

[5] Aus Platzgründen kann hier nur ein kurzer Überblick über die Datenstrukturen gegeben werden. Für eine umfassende Darstellung sei auf [UDDI] verwiesen.

[6] Die Struktur der UUIDs und der Erzeugungsalgorithmus sind im Standard `ISO/IEC 11578:1996` beschrieben.

Die zentralen Datenstrukturen businessEntity, businessService und bindingTemplate sind logisch in einer Baumstruktur angeordnet. Eine businessEntity kann mehrere registrierte businessServices haben, die in einer Vater/Sohn-Beziehung zur businessEntity stehen. Analog können einem businessService mehrere bindingTemplates zugeordnet sein. Die tModels stehen in keiner direkten Hierarchiebeziehung zu den anderen Datenstrukturen, sondern werden von diesen zu Klassifikationszwecken genutzt. Der Abschnitt „Einsatz von tModels“ widmet sich dieser Datenstruktur im Detail. Durch publisherAssertions können Geschäftsbeziehungen wie etwa Allianzen, Partnerschaften oder Firmenbeziehungen wie Muttergesellschaft/Tochtergesellschaft beschrieben werden.

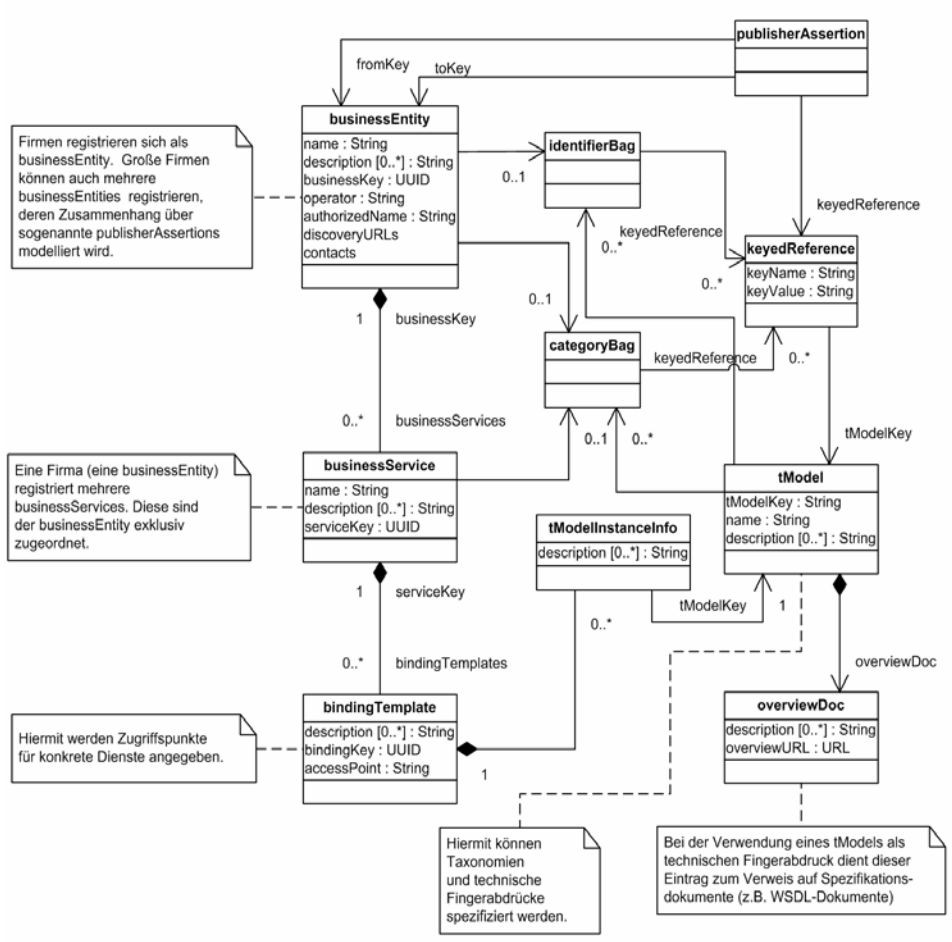


Abbildung 10.5: UML-Modell des UDDI-Schemas

Wie sieht nun im praktischen Einsatz die Verwendung eines UDDI-Verzeichnisdienstes aus Sicht eines Dienstanbieters aus? Ein Anbieter registriert zunächst eine businessEntity. Dabei werden Angaben zu Namen, Adressen und Kontaktpersonen gemacht. Zusätzlich können verschiedene Taxonomien zur Identifikation (z. B. D-U-N-S[7] Nummern) sowie zur Kategorisierung des Eintrags verwendet werden (z. B. `ISO 3166` zur geographischen Kategorisierung). Die im Folgenden angegebenen XML-Dokumente können zur Registrierung bei UDDI-Servern verwendet werden (die konkreten SOAP-Nachrichten zur Registrierung folgen später), wobei eine Besonderheit zu beachten ist: Die hier aus Gründen der Vollständigkeit angegebenen UUIDs (xxKey-Attribute) dürfen bei der Registrierung nicht mit angegeben werden, da sie automatisch bei der Erstregistrierung durch den UDDI-Server vergeben werden. Bei späteren Änderungen dieser Daten bleiben jedoch die UUIDs unverändert, so dass sie zur Referenzierung der Elemente verwendet werden können.

```
<?xml version="1.0" encoding="UTF-8" ?>
<businessEntity  businessKey="35C43D08-FBBF-2C59-AA29-CF032D054446">
  <name>Universitaet Passau - Lehrstuhl fuer Dialogorientierte Systeme</name>
  <description>Beispiel - Anwendung fuer Dienste rund um das Wetter</description>
  <categoryBag>
    <keyedReference  keyName="uddi-org:iso-ch:3166:1999" keyValue="DE-BY-PAS"
                     tModelKey="uuid:61668105-B6B6-425C-914B-409FB252C36D" />
  </categoryBag>
</businessEntity>
```

XML-Dokument 10.1: businessEntity

XML-Dokument 10.1 zeigt einen businessEntity-Eintrag, der für die Registrierung einer Organisation, hier Universitaet Passau, verwendet wurde. Der businessKey wurde bei der Registrierung automatisch vergeben. Als Kategorisierungsinformation ist angegeben, dass diese Organisation gemäß ISO-Klassifikation in Deutschland-Bayern-Passau (DE-BY-PAS) liegt. Kategorisierungsinformationen werden stets mit Hilfe von tModels innerhalb von keyedReference-Elementen mit den Attributen tModelKey, keyValue und keyName angegeben. Wie Abbildung 10.5 zeigt, werden keyedReferences eingesetzt, um Klassifizierungen mit Hilfe von tModels auszudrücken. Das verwendete tModel (angegeben im Attribut tModelKey) bezeichnet hier das abstrakte Konzept der geographischen ISO-Klassifizierung. Die Kategorie, der die businessEntity zugeordnet ist, steht im Attribut keyValue und muss im Rahmen der angegebenen Klassifizierung interpretiert werden.

---

[7] Dun & Bradstreet Number (`www.dnb.com`): Weltweiter Identifikator für Unternehmen.

Hat ein Anbieter eine businessEntity registriert, folgt als nächster Schritt die Angabe der Dienste. Die verschiedenen Arten von angebotenen Diensten werden unter verschiedenen businessServices zusammengefasst. In unserem Temperatur-Beispiel stellen der Temperaturdienst und der Einheitenumrechnerdienst zwei verschiedene businessServices dar. Im Gegensatz dazu sind der TemperaturInFahrenheitDienst und der Temperaturdienst dem gleichen businessService zugeordnet. Registriert der Dienstanbieter einen businessService, muss der businessKey der zugehörigen Firma angegeben werden, um eine eindeutige Zuordnung herstellen zu können. Das XML-Dokument 10.2 zeigt einen solchen businessService.

```
<?xml version="1.0" encoding="UTF-8" ?>
<businessService  serviceKey="362C66B3-03C4-F54B-AC4F-CDC8E2872EED"
                  businessKey="35C43D08-FBBF-2C59-AA29-CF032D054446">
  <name>Temperaturauskunft</name>
  <description>Der Dienst liefert die Temperatur an einem gegebenen Datum</description>
  <categoryBag>
    <keyedReference  keyName="uddi-org:iso-ch:3166:1999" keyValue="DE-BY-PAS"
                     tModelKey="uuid:61668105-B6B6-425C-914B-409FB252C36D" />
  </categoryBag>
</businessService>
```

XML-Dokument 10.2: businessService

Wie man sieht stellt ein businessService lediglich eine Dienstgruppierung dar. Es wird noch keinerlei Auskunft über konkrete Dienste gegeben. Durch die Registrierung von businessEntity und businessService ist nun der Rahmen geschaffen, um konkrete Dienste registrieren zu können. Durch das Hinzufügen von Daten über konkrete Dienste mittels der Datenstruktur bindingTemplate werden Dienste an businessServices gebunden.

```
<?xml version="1.0" encoding="UTF-8"?>
<bindingTemplate  bindingKey='"4FB831F6-327E-7815-3331-52DB9BF9C220"
                  serviceKey="362C66B3-03C4-F54B-AC4F-CDC8E2872EED">
  <description>Temperaturangabe Passau in Grad Celsius</description>
  <accessPoint URLType="http">http://wetter.fmi.uni-passau.de:8004/service/Temperaturdienst</accessPoint>
  <tModelInstanceDetails>
    <tModelInstanceInfo tModelKey="uuid:B39997F5-DF7F-9F62-0EEE-6F43345DE1A3"/>
  </tModelInstanceDetails>
</bindingTemplate>
```

XML-Dokument 10.3: bindingTemplate

Das XML-Dokument 10.3 zeigt ein Dokument, das zur Registrierung des Temperaturdienstes verwendet werden kann. Der Dienst kann via HTTP über die angegebene URL angesprochen werden (URLType="http"). UDDI unterstützt neben HTTP noch andere Typen von Zugriffspunkten wie Fax, Email, etc. In einem bindingTemplate ist jedoch explizit noch keine Beschreibung konkreter Aufruf- und Rückgabemodalitäten (z. B.

Signaturen) enthalten. Diese finden sich erst im tModel, welches durch den angegebenen tModelKey referenziert wird (siehe nächster Abschnitt).

Um Beziehungen zwischen Unternehmen zu beschreiben (z. B. Muttergesellschaft/Tochtergesellschaft, Allianzen, Partnerschaften), können publisherAssertions verwendet werden. Registriert eine Firma eine publisherAssertion, bleibt diese so lange für andere unsichtbar, bis der andere beteiligte Partner dieselbe publisherAssertion registriert. Erst dann wird diese gültig und öffentlich sichtbar. Durch diese symmetrische Registrierung kann Missbrauch weitestgehend verhindert werden.

Einsatz von tModels

Prinzipiell gibt es in UDDI zwei verschiedene Anwendungsszenarien für tModels:

1. tModel als technischer Fingerabdruck und
2. tModel als Namensraumbezeichner.

Dient ein tModel als technischer Fingerabdruck, wird es nur in der bindingTemplate-Datenstruktur verwendet. Dort referenziert es technische Dienstbeschreibungen, d. h., es verweist auf Spezifikationsdokumente. Als Namensraumbezeichner finden tModels unter anderem Verwendung in identifierBags, categoryBags und puplisherAssertions.

Technischer Fingerabdruck

Im XML-Dokument 10.3 wurde bereits gezeigt, dass sich in einem bindingTemplate ein Zugriffspunkt zu einem angebotenen Dienst befindet. Allerdings fehlten detaillierte Informationen für einen Anwendungsentwickler wie etwa Aufruf- und Rückgabeparameter, verwendete Transportprotokolle, etc. Zur Beschreibung genau dieser Informationen wurde WSDL entwickelt und die UDDI-Initiative hat deshalb eine Empfehlung herausgegeben (als so genanntes „best practices“-Dokument [CuER01]), wie WSDL in UDDI eingesetzt werden kann, um diese Informationen anzugeben (siehe auch Abschnitt 10.5.3). Laut dieser Empfehlung sollen die Informationen nicht direkt in einer UDDI-Datenstruktur abgelegt werden, sondern in einem separaten WSDL-Dokument, das von einem tModel referenziert wird. Das in diesem Kapitel beschriebene Beispiel eines Temperaturdienstes folgt dieser Empfehlung.

```
<tModel tModelKey="uuid:B39997F5-DF7F-9F62-0EEE-6F43345DE1A3">
  <name>Temperaturauskunft</name>
  <description xml:lang="de">Die technische Spezifikation eines Temperaturdienstes</description>
  <overviewDoc>
    <description xml:lang="de">WSDL Dokument</description>
    <overviewURL>http://wetter.fmi.uni-passau.de/Temperaturdienst.wsdl</overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference  tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
                     keyName="uddi-org:types" keyValue="wsdlSpec"/>
  </categoryBag>
</tModel>
```

XML-Dokument 10.4: tModel für Temperaturdienst

Im XML-Dokument 10.4 fällt auf, dass die Art eines tModels (hier: tModel, das auf ein WSDL-Dokument verweist) durch ein generisches UDDI-Klassifikations-tModel (uuid:C1ACF…) angegeben wird. Es fällt ebenfalls auf, dass sich die UUIDs eines tModels von den UUIDs anderer UDDI-Datenstrukturen durch ein vorangestelltes uuid: unterscheiden.

Namensraumbezeichner

Die zweite Anwendungsmöglichkeit von tModels ist wesentlich allgemeiner als die eines technischen Fingerabdrucks. tModels können verwendet werden, um Taxonomien zu erzeugen und zu benutzen. Somit können eigene, anwendungsspezifische Klassifizierungen definiert und angewendet werden, oder auch weltweit etablierte Standard-Klassifizierungen verwendet werden.

Beispiele für Standard-Klassifizierungen, die hauptsächlich bei businessEntities und businessServices Anwendung finden, sind: North American Industry Classification System (NAICS), Universal Standard Products and Services Classification (UNSPSC) oder ISO 3166 als internationaler Standard für geographische Regionen.

Geprüfte und ungeprüfte Taxonomien

Um eine global konsistente und korrekte Verwendung von Taxonomien mit zugehörigen Schlüssel/Wert-Paaren sicherzustellen, können in UDDI keyedReference-Einträge, die in identifierBags bzw. categoryBags auftreten, durch einen externen Validierungsdienst überprüft werden. Natürlich können ebenfalls Dokumente registriert werden, die beliebige, ungeprüfte Taxonomien verwenden. Insbesondere steht es jedem Anbieter frei, eigene tModels zur Erzeugung von Taxonomien zu definieren und diese dann ungeprüft verwenden zu lassen.

Anfragen

Nachdem in den vorangegangenen Abschnitten die Datenstrukturen vorgestellt wurden, die in einem UDDI-Verzeichnisdienst Verwendung finden, wird im Folgenden auf die zweite wesentliche Komponente des UDDI-„Datenbanksystems“ eingegangen: die Anfragesprache. Generell finden alle Anfragen in einer XML-basierten Anfragesprache statt. Sowohl die Anfragen als auch deren Ergebnisse werden über das SOAP-Protokoll zwischen UDDI-Server und Klient übertragen. UDDI unterstützt nur einen sehr kleinen Teil der vollen Funktionalität des SOAP-Protokolls, beispielsweise werden keine SOAP-Header-Elemente unterstützt.

Insgesamt definiert Version 2.0 von UDDI eine API mit etwa 25 Anfragen und 15 Antwortdokumenten.[8] Die UDDI-Spezifikation beschränkt sich bewusst auf relativ einfache Anfragetypen (beispielsweise sind keine Joins möglich) und überlässt es höheren Schichten wie etwa Suchmaschinen oder Marktplätzen, komplexere Anfragen durch Middleware-Funktionalität zu unterstützen. Die Anfragesprache von UDDI umfasst lediglich grundlegende Operationen zum Einfügen, Ändern, Löschen und Auffinden von Daten.

Prinzipiell lässt sich die API aufteilen in einen Teil zur Datenabfrage (Inquiry-API) und einen Teil zur Datenmodifikation (Publishing-API).

Inquiry-API

Innerhalb dieser API können Browse-Anfragen, die zum erstmaligen Auffinden von Metadaten dienen, und Drill-Down-Anfragen, die vollständige und detaillierte Daten zurückliefern, unterschieden werden. Auf diese API greift zum Beispiel ein UDDI-Browser zurück, der Klienten eine interaktive Dienstsuche bietet.

Browse-Anfragen

Anfragen dieses Typs beginnen mit find_xx. Anhand der Anfrage aus XML-Dokument 10.5 werden die Möglichkeiten dieses Anfragetyps verdeutlicht.

---

[8] Im Rahmen dieses Abschnitts können nur einige Anfragebeispiele gezeigt werden. Für eine umfassende Referenz sei auf [UDDI] verwiesen.

```
<find_business maxRows="5" generic="2.0" xmlns="urn:uddiorg:api_v2">
    <findQualifiers>
      <findQualifier>sortByNameAsc</findQualifier>
    </findQualifiers>
    <name>Universitaet</name>
</find_business>
```

XML-Dokument 10.5: Syntax einer find_business-Anfrage

Mit dieser Anfrage kann nach businessEntities gesucht werden, deren Name mit Universitaet beginnt.[9] Zusätzlich wurde spezifiziert, dass höchstens fünf Treffer zurückgeliefert werden (maxRows="5") und dass die Ergebnisse aufsteigend nach Namen sortiert sein sollen (sortByNameAsc). Als Ergebnis liefert diese Anfrage eine Liste von businessInfo-Elementen. Diese enthalten neben businessEntity-Daten auch Informationen über registrierte businessServices der gefundenen Firmen. Es existiert noch eine Reihe von weiteren möglichen Selektionskriterien. Beispielsweise kann eine Einschränkung auf Firmen erfolgen, die Dienste anbieten, die bestimmten tModels genügen.

Drill-Down-Anfragen

Dieser Anfragetyp wird verwendet, wenn man einen Identifikator bereits kennt und die dazugehörigen Detaildaten abfragen will. Diese Anfragen beginnen mit get_xx. Als konkretes Beispiel dient die get_tModelDetail-Anfrage aus XML-Dokument 10.6. Sie liefert die Informationen über ein tModel eingebettet in ein tModelDetail-Element zurück.

```
<get_tModelDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
  <tModelKey>uuid:B39997F5-DF7F-9F62-0EEE-6F43345DE1A3</tModelKey>
</get_tModelDetail>
```

XML-Dokument 10.6: Syntax einer get_tModelDetail-Anfrage

Publishing-API

Neben den Anfragen, durch die Metadaten abgefragt werden können, gibt es entsprechende Aufrufe um Metadaten registrieren, verändern und löschen zu können. Diese beginnen mit save_xx zur Registrierung bzw. zur Änderung und mit delete_xx zum Löschen von Daten. Aus Sicherheitsgründen arbeiten alle Aufrufe dieser Publishing-API mit verschlüsselten Verbindungen zum UDDI-Server (via HTTPS mit SSL 3.0) und der Klient muss sich durch ein Authentifizierungstoken ausweisen. Dieses Token erhält der Klient durch Aufruf von

[9] Die UDDI-Spezifikation legt fest, dass ein Präfixvergleich durchgeführt wird, wenn bei der Suche keine Wildcards verwendet werden.

get_authToken vom UDDI-Server. Der save_business-Aufruf aus XML-Dokument 10.7 kann zur Registrierung einer businessEntity verwendet werden.

```
<save_business generic="2.0" xmlns="urn:uddi-org:api_v2" >
  <authInfo>xy_token</authInfo>
  <businessEntity businessKey="">
    <name>Universitaet Passau - Lehrstuhl fuer Dialogorientierte Systeme</name>
    <description>Beispiel - Anwendung fuer Dienste rund um das Wetter</description>
    <!-- Hier können weitere Daten zu businessServices, bindingTemplates, etc. stehen -->
  </businessEntity>
</save_business>
```

XML-Dokument 10.7: save_business-Aufruf

Neben dem authInfo-Element für das Authentifizierungstoken besitzt der Aufruf ein businessEntity-Element. Darin können die relevanten Informationen (siehe UML-Diagramm) registriert werden, wobei zu beachten ist, dass bei der Erstregistrierung ein leerer businessKey (businessKey="") übergeben werden muss. Der Schlüssel selbst wird vom UDDI-Server generiert. Gleichzeitig können hierbei als Unterelemente des businessEntity-Eintrags auch businessServices und bindingTemplates registriert werden. Auch bei diesen bedeutet die Übergabe eines leeren Schlüssels die Neuregistrierung. Werden stattdessen bestehende Schlüssel verwendet, werden dadurch bestehende Einträge geändert.

Pufferung und Puffer-Kohärenz

Die UDDI-Initiative schlägt für den Umgang mit bindingTemplate-Informationen ein spezielles Vorgehen vor. Dieses Vorgehen wird anhand des folgenden Beispiels beschrieben: Eine Anwendung A, welche den Temperaturdienst T benutzen will (Die UUID von T muss bereits zur Entwicklungszeit von A bekannt sein.) geht folgendermaßen vor:

1. Um T erstmalig aufrufen zu können, müssen die Metadaten von T mit get_bindingTemplate vom UDDI-Server abgerufen werden. Die dabei erhaltenen technischen Informationen (insbesondere der Zugriffspunkt) können von A lokal gepuffert werden, um die Belastung des UDDI-Servers gering zu halten, beispielsweise wenn A den Dienst T mehrfach benötigt.
2. Schlägt ein Aufruf von T fehl (tritt beispielsweise ein HTTP-404-Fehler auf), muss A erneut mittels einer get_bindingDetail-Anfrage die (hoffentlich) aktualisierten Metadaten von T vom UDDI-Server abrufen. Unterscheiden sich diese von der gepufferten (alten) Version, sollen die neuen Metadaten gepuffert und die alte Version aus dem lokalen Cache entfernt werden.

Damit will die UDDI-Initiative einerseits einer Überlastung der UDDI-Server vorbeugen, indem es eine lokale Pufferung der Anfrageergebnisse empfiehlt, andererseits aber einer Überalterung und damit einer Unbrauchbarkeit der gepufferten Daten vorbeugen, indem im Falle eines Fehlers eine erneute Anfrage beim UDDI-Server erfolgt. Aus Datenbanksicht beschreibt dieses Vorgehen ein Verfahren um Puffer-Kohärenz zu erzielen.

Replikation

Nachdem bereits die Datenstrukturen und die Anfragesprache des UDDI-„Datenbanksystems“ vorgestellt wurden, soll hier kurz auf das Replikationsverfahren in UDDI eingegangen werden. Die Motivation für die Definition eines globalen, verteilten Verzeichnisdienstes ist offensichtlich: Die Verwendung von UDDI ist umso attraktiver, je größer die Datenbasis des UDDI-Servers ist. Umfasst diese beispielsweise nur deutsche Dienste entgehen dem Benutzer unter Umständen alternative Dienste aus Europa. Aus diesem Grund wird ein weltweiter Verzeichnisdienst (UDDI-Wolke) angestrebt, der verschiedene Instanzen von UDDI-Servern besitzt, dessen Gesamtdaten sich jedoch über jede Instanz abfragen lassen. Der Datenabgleich zwischen den Servern soll durch eine Replikationsrichtlinie erreicht werden. In der Datenbankliteratur [Dada96] findet sich eine Vielzahl von Ansätzen zur Replikation und Allokation von Daten. Speziell für Metadaten-Server-Verbunde wurden auch Ansätze entwickelte, die auf Publish-Subscribe-Architekturen beruhen und versuchen, den erforderlichen Kommunikationsaufwand zu reduzieren, z. B. [KKKK01, KKKK02].

Die UDDI-Initiative schlägt zum Datenabgleich zwischen den UDDI-Servern einen Primärkopie-Ansatz vor, der darauf beruht, dass jedes Datum im Verzeichnis einen Heimat-UDDI-Server besitzt, an dem es registriert wurde und an dem es geändert bzw. gelöscht werden kann. Die Daten werden gewissermaßen an diesem Heimat-Server verwahrt. Periodisch werden die anderen UDDI-Server benachrichtigt und ihnen wird der aktuelle Replikationsstatus des meldenden Servers mitgeteilt. Stellt ein Server fest, dass er veraltete Daten besitzt, weil seit dem letzten Abgleich auf einem anderen Server Änderungen vorgenommen wurden, fordert er die entsprechenden Änderungsdatensätze an. Änderungen werden dabei mit Hilfe von global eindeutigen, aufsteigenden Sequenznummern erkannt. Die gesamte zur Replikation erforderliche Kommunikation erfolgt via SOAP auf verschlüsselten HTTP-Verbindungen.

Die in den vorangegangenen Abschnitten beschriebenen Strukturen und

Eigenschaften von UDDI beziehen sich auf Version 2 von UDDI. Die Version 3 des UDDI-Standards wird Spezifikationen zur Authentifizierung, zur Autorisierung, zur Versionskontrolle und zur Historisierung von Änderungen enthalten.

10.4.2. Dienstsuche: WS-Inspection

Wie bereits erwähnt ist ein UDDI-Verzeichnis nicht die einzige Möglichkeit, Dienste auffindbar zu machen. Ein Ansatz, der nicht als Konkurrenz, sondern als Ergänzung zu UDDI und anderen Verfahren gedacht ist, ist die Web Services Inspection Language (WS-Inspection). Ein WS-Inspection-Dokument ist im Wesentlichen eine Sammlung von Verweisen auf bereits existierende Dokumente, die Web Services beschreiben. Damit bietet WS-Inspection die Möglichkeit, eine Sammlung aller Referenzen auf Beschreibungen von Diensten dort verfügbar zu machen, wo die Dienste auch angeboten werden, ohne dass die Beschreibungen repliziert werden müssen. Ein WS-Inspection-Dokument kann prinzipiell Verweise auf beliebige Dokumente beinhalten, sieht allerdings spezielle Elemente für Verweise auf Dokumente, die in UDDI-Verzeichnissen gespeichert sind, und für WSDL-Dokumente vor und ist durch seinen Erweiterungsmechanismus flexibel erweiterbar. Wir werden uns in diesem Kapitel allerdings auf die wichtigsten Möglichkeiten von WS-Inspection in Bezug auf UDDI und WSDL konzentrieren.

```
<?xml version="1.0" encoding="UTF-8"?>
<inspection   xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
              xmlns:wsiluddi="http://schemas.xmlsoap.org/ws/2001/10/inspection/uddiv2/">
  <service>
    <abstract>Der Dienst liefert die Temperatur an einem gegebenen Datum.</abstract>
    <name>Temperaturdienst</name>
    <description  referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
                  location="http://wetter.fmi.uni-passau.de/Temperaturdienst.wsdl" />
    <description referencedNamespace="urn:uddi-org:api_v2">
      <wsiluddi:serviceDescription location="http://wetter.fmi.uni-passau.de:9004/uddi/inquiryapi/uddi">
        <wsiluddi:serviceKey>362C66B3-03C4-F54B-AC4F-CDC8E2872EED</wsiluddi:serviceKey>
      </wsiluddi:serviceDescription>
    </description>
  </service>

  <service>
    <abstract>Der Dienst rechnet Temperaturangaben in verschiedene Einheiten um.</abstract>
    <name>Einheitenumrechnerdienst</name>
    <description  referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
                  location="http://wetter.fmi.uni-passau.de/Einheitenumrechnerdienst.wsdl" />
    <description referencedNamespace="urn:uddi-org:api_v2">
      <wsiluddi:serviceDescription location="http://wetter.fmi.uni-passau.de:9004/uddi/inquiryapi/uddi">
        <wsiluddi:serviceKey>2C78DBF9-4295-FBAC-7CC7-62F9609AB865</wsiluddi:serviceKey>
      </wsiluddi:serviceDescription>
    </description>
  </service>
```

```
  <service>
    <abstract>Der Dienst liefert die Temp. an einem gegebenen Datum in Grad Fahrenheit.</abstract>
    <name>TemperaturInFahrenheitDienst</name>
    <description  referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
                  location="http://wetter.fmi.uni-passau.de/TemperaturInFahrenheitDienst.wsdl" />
    <description referencedNamespace="urn:uddi-org:api_v2">
      <wsiluddi:serviceDescription location="http://wetter.fmi.uni-passau.de:9004/uddi/inquiryapi/uddi">
        <wsiluddi:serviceKey>362C66B3-03C4-F54B-AC4F-CDC8E2872EED</wsiluddi:serviceKey>
      </wsiluddi:serviceDescription>
    </description>
  </service>
</inspection>
```

WS-Inspection-Dokument 10.1: Dienste von wetter.fmi.uni-passau.de

WS-Inspection-Dokument 10.1 zeigt die Dienste eines Anbieters am Beispiel unseres Servers `wetter.fmi.uni-passau.de`. Dieser Server bietet drei Dienste an: den Temperaturdienst, den Einheitenumrechnerdienst und den TemperaturInFahrenheitDienst. Zu allen drei Diensten liegen sowohl entsprechende WSDL-Dokumente als auch Einträge in einem UDDI-Verzeichnisdienst vor. Ein WS-Inspection-Dokument hat einen sehr einfachen Aufbau: das Wurzelelement heißt inspection und beinhaltet Verweise auf Dienstbeschreibungen oder Verweise auf andere WS-Inspection-Dokumente als Unterelemente. Für jeden Dienst, der in dem WS-Inspection-Dokument aufgeführt ist, existiert ein eigenes service-Element. Wir werden die Bedeutung der verschiedenen Unterelemente eines service-Elementes am Beispiel des Temperaturdienstes beschreiben. Das erste Element im Beispiel ist das abstract-Element. Dieses Element ist optional, beinhaltet eine kurze textuelle Beschreibung des Dienstes und ist, genau wie das nachfolgende name-Element, für menschliche Leser und nicht zur maschinellen Auswertung gedacht. Das optionale name-Element gibt den Namen des Dienstes an. Danach folgen ein oder mehrere description-Elemente, die jeweils einen Verweis auf eine Beschreibung des Dienstes beinhalten. In dem Beispieldokument sind das jeweils ein Verweis auf UDDI und ein WSDL-Dokument. Zuerst erklären wir den Verweis auf das WSDL-Dokument:

```
<description  referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
              location="http://wetter.fmi.uni-passau.de/Temperaturdienst.wsdl" />
```

Das Attribut referencedNamespace bezeichnet den Namensraum, zu dem das referenzierte Dokument gehört. In unserem Fall handelt es sich um ein WSDL-Dokument. Dieser Namensraumbezeichner kann von einem Benutzer des WS-Inspection-Dokumentes genutzt werden, um festzustellen, ob er mit dem angegebenen Dokument umgehen kann und es deswegen wert ist geladen zu werden oder nicht. Das Attribut location

spezifiziert eine URL, die genutzt werden kann, um das referenzierte Dokument zu laden. Dieses Attribut muss nicht angegeben werden, wenn man den Erweiterungsmechanismus von WS-Inspection nutzt. Für unser Beispiel reicht allerdings dieser Standardansatz aus, um auf das WSDL-Dokument zu verweisen. Die Referenz auf eine Beschreibung, die in einem UDDI-Verzeichnisdienst gespeichert ist, demonstriert den Einsatz des Erweiterungsmechanismus am Beispiel der standardisierten UDDI-Erweiterung. Hier legt das Attribut referencedNamespace fest, dass die Beschreibung in einem UDDI-Verzeichnis liegt, das die Protokollversion 2 versteht. Wie man zu der Beschreibung gelangt, ist in diesem Fall durch Unterelemente festgelegt: Das location-Attribut des serviceDescription-Elementes spezifiziert die URL, unter der die Inquiry-API des UDDI-Verzeichnisdienstes angesprochen werden kann. Der Wert des serviceKey-Elementes kann dann dazu verwendet werden, um an den UDDI-Verzeichnisdienst eine get_serviceDetail-Nachricht via SOAP zu schicken und die gewünschten Informationen zu erhalten.

Die Möglichkeiten von WS-Inspection gehen über die hier gezeigten hinaus. Zum einen ist WS-Inspection flexibel erweiterbar, zum anderen erlaubt es, Hierarchien von WS-Inspection-Dokumenten aufzubauen. Diese Möglichkeit ist für größere Firmen mit mehreren Abteilungen sehr wichtig, da damit ein zentrales WS-Inspection-Dokument der Firma nur auf die Dokumente der Abteilungen verweisen muss und die Abteilungen ihre eigenen WS-Inspection-Dokumente selbst verwalten können. Außerdem kann innerhalb von WS-Inspection-Dokumenten auch auf businessEntity-Einträge in UDDI-Verzeichnisdiensten verwiesen werden, statt auf einzelne businessService-Einträge wie im Beispiel.

Nachdem wir das Format von WS-Inspection-Dokumenten vorgestellt haben, bleibt noch die Frage zu klären, wie WS-Inspection-Dokumente gefunden werden können. Dazu werden im Standard zwei Möglichkeiten festgelegt: Zum einen wird vorgeschlagen, WS-Inspection-Dokumente unter dem Namen „inspection.wsil“ an den Hauptseiten des Web-Servers des Anbieters abzulegen. In unserem Fall ist das WS-Inspection-Dokument unseres Servers also unter der Adresse `http://wetter.fmi.uni-passau.de/inspection.wsil` abrufbar. Die zweite Möglichkeit besteht darin, von HTML-Seiten aus mit Hilfe eines META-Elementes auf WS-Inspection-Dokumente zu verweisen. Diese Möglichkeit wird im HTML-Dokument 10.1 demonstriert. Ein geeigneter Browser kann den Benutzer dann auf vorhandene WS-Inspection-Dokumente hinweisen.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//DE">
<HTML>
  <HEAD>
    <TITLE>Web-Dienste am Beispiel eines Temperaturdienstes</TITLE>
    <META name="serviceInspection" content="http://wetter.fmi.uni-passau.de/inspection.wsil">
  </HEAD>

  <BODY>
    <!-- … Text der HTML-Seite … -->
  </BODY>
</HTML>
```

HTML-Dokument 10.1: Verweis auf ein WS-Inspection-Dokument in einer HTML-Seite

10.5. Dienstbeschreibung (WSDL)

Das Finden eines Dienstes mit Hilfe von UDDI oder WS-Inspection reicht nicht aus, um diesen Dienst auch aufrufen zu können. Dazu benötigt man die genauen technischen Spezifikationen, die festlegen, welche Operationen der Dienst unterstützt, wie das Format der Eingabe- und Ausgabedaten beschaffen sein muss und wie die Übertragung der Daten zu geschehen hat, d. h. über welches Protokoll mit dem Dienst kommuniziert wird. Genau dafür wurde die Web Services Description Language (WSDL) [CCMW01] von Ariba, IBM und Microsoft entwickelt und beim W3C zur Standardisierung eingereicht.

Die WSDL-Spezifikation legt fest, auf welche Weise man die zum Aufruf eines Dienstes notwendigen Informationen in einem XML-Dokument ablegt. Im Folgenden werden wir die einzelnen Komponenten eines WSDL-Dokuments am Beispiel des Temperaturdienstes vorstellen.

10.5.1. Struktur eines WSDL-Dokumentes

WSDL verwendet zur Beschreibung von Diensten sechs verschiedene Elemente: types, message, portType, binding, port und service (siehe Abbildung 10.6). WSDL betrachtet einen Dienst (service) als eine Menge von (abstrakten) Endpunkten von Netzwerkverbindungen. Diese Endpunkte tauschen Nachrichten (messages) untereinander aus, wobei eine Nachricht eine abstrakte Beschreibung der ausgetauschten Daten darstellt. Operationen repräsentieren abstrakte Beschreibungen möglicher Aktionen eines Dienstes und bestehen prinzipiell aus Ein- und Ausgabenachrichten. Operationen werden zu Port-Typen (portTypes) zusammengefasst. Dabei handelt es sich im Grunde um eine Menge von Operationen, die von einem oder mehreren Endpunkten unterstützt

werden. Bis hierhin sind die jeweiligen Definitionen unabhängig von bestimmten Netzwerkprotokollen und Datenformaten. Erst eine so genannte Bindung (binding) legt in WSDL für einen Port-Typ ein bestimmtes Netzwerkprotokoll und Datenformat fest. Ein Port (port) fasst anschließend eine Netzwerkadresse mit einer Bindung zusammen. Ein Dienst selbst besteht dann aus einer Menge von (zusammengehörigen) Ports.

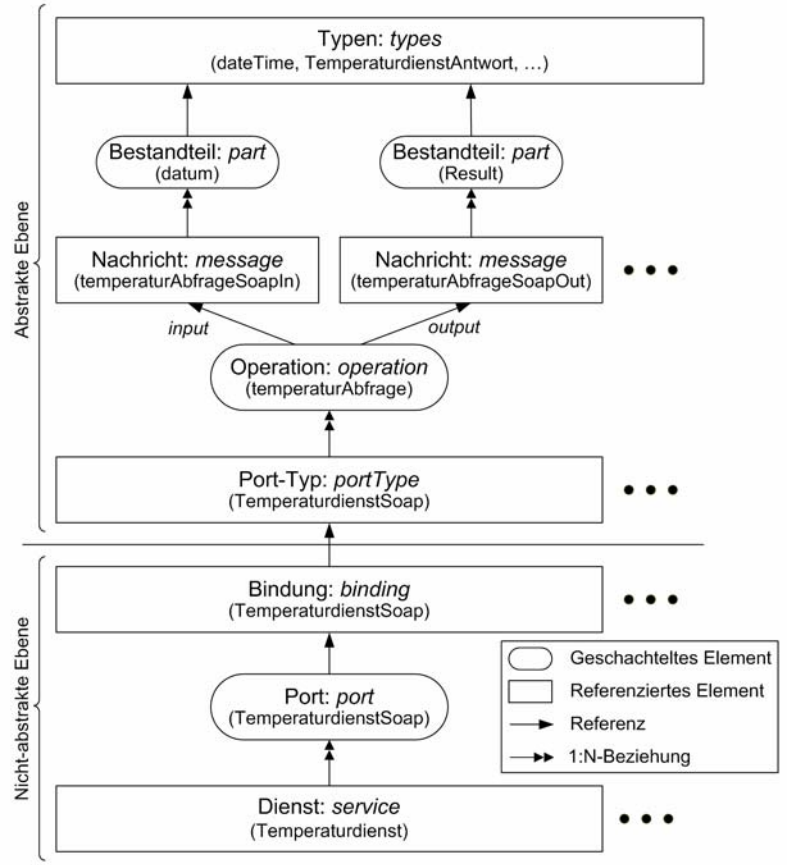


Abbildung 10.6: Struktureller Aufbau eines WSDL-Dokumentes

Ein WSDL-Dokument besteht prinzipiell aus einer Menge von Definitionen. Jede Definition wird durch ein eigenes Element im WSDL-Dokument repräsentiert. Jedes dieser Elemente muss ein Unterelement des Wurzelelements definitions sein. Als vollständiges Beispiel zeigt WSDL-Dokument 10.1, wie der Temperaturdienst in WSDL beschrieben wird. Abbildung 10.6 zeigt den strukturellen Aufbau eines WSDL-Dokuments am Beispiel des Temperaturdienstes aus WSDL-Dokument 10.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions   name="Temperaturdienst"
               targetNamespace="http://wetter.fmi.uni-passau.de/Temperaturdienst.wsdl"
               xmlns:tns="http://wetter.fmi.uni-passau.de/Temperaturdienst.wsdl"
               xmlns:ns1="http://wetter.fmi.uni-passau.de/Temperaturdienst.xsd"
               xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema    xmlns="http://www.w3.org/2001/XMLSchema"
               targetNamespace="http://wetter.fmi.uni-passau.de/Temperaturdienst.xsd">
      <complexType name="TemperaturdienstAntwort">
        <sequence>
          <element name="temperatur" type="double"/>
          <element name="einheit" type="string"/>
          <element name="messdatum" type="dateTime"/>
        </sequence>
      </complexType>
    </schema>
  </types>

  <message name="temperaturAbfrageSoapIn">
    <part name="datum" type="xsd:dateTime"/>
  </message>
  <message name="temperaturAbfrageSoapOut">
    <part name="Result" type="ns1:TemperaturdienstAntwort"/>
  </message>

  <portType name="TemperaturdienstSoap">
    <operation name="temperaturAbfrage" parameterOrder="datum">
      <input name="temperaturAbfrageSoapIn" message="tns:temperaturAbfrageSoapIn"/>
      <output name="temperaturAbfrageSoapOut" message="tns:temperaturAbfrageSoapOut"/>
    </operation>
  </portType>

  <binding name="TemperaturdienstSoap" type="tns:TemperaturdienstSoap">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="temperaturAbfrage">
      <soap:operation soapAction="temperaturAbfrage" style="rpc"/>
      <input name="temperaturAbfrageSoapIn">
        <soap:body   use="encoded" namespace="http://wetter.fmi.uni-passau.de/Temperaturdienst"
                     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output name="temperaturAbfrageSoapOut">
        <soap:body   use="encoded" namespace="http://wetter.fmi.uni-passau.de/Temperaturdienst"
                     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>

  <service name="Temperaturdienst">
    <port name="TemperaturdienstSoap" binding="tns:TemperaturdienstSoap">
      <soap:address location="http://wetter.fmi.uni-passau.de:8004/service/Temperaturdienst"/>
    </port>
  </service>

</definitions>
```

WSDL-Dokument 10.1: Beschreibung des Temperaturdienstes

WSDL erlaubt es, die Beschreibung von Diensten zu modularisieren. Dazu lagert man einzelne, zusammengehörende Teile in eigene Dateien aus und importiert diese dann mit einem import-Element in das Hauptdokument:[10]

```
<definitions …>
  <import namespace="uri" location="uri"/> *
</definitions>
```

Zum Beispiel kann man die Definition der Datentypen von Nachrichten in eine eigene Datei auslagern und mit Hilfe eines solchen Elements in das WSDL-Dokument einbinden.

Bei der Definition des WSDL-Standards wurde darauf geachtet Erweiterbarkeit zu gewährleisten. Dies ist insbesondere bei der Bindung von abstrakten Diensten an bestimmte Protokolle und Datenformate wichtig. WSDL erlaubt die Angabe von so genannten Erweiterungselementen als Unterelemente von bestimmten WSDL-Elementen. Mit ihnen können etwa technische Spezifikationen für eine Bindung angegeben werden, da diese im Allgemeinen vom verwendeten Protokoll und Datenformat abhängen.

Im Folgenden werden nun die einzelnen Definitionen, d. h. die Unterelemente des definitions-Wurzelelements eines WSDL-Dokuments, beschrieben. Zuerst werden die Elemente beschrieben, die zur Beschreibung der abstrakten Teile von Diensten verwendet werden. Die Beschreibung der Elemente für die nicht-abstrakten Teile folgt in Abschnitt 10.5.2.

Typen

Das types-Element dient zur Definition von Datentypen, die in den ausgetauschten Nachrichten verwendet werden. Die Syntax sieht folgendermaßen aus:

```
<definitions …>
  <types>
    <xsd:schema … /> *
  </types>
</definitions>
```

[10] Zur Beschreibung der XML-Grammatik wird die gleiche informelle Syntax benutzt wie in der WSDL-Spezifikation. Ein * hinter einem Element bedeutet, dass es beliebig oft (auch 0 mal) vorkommen kann, ein ?, dass es 0 oder 1 mal vorkommen kann.

Aus Gründen der Interoperabilität wird die Verwendung von XML-Schema zur Definition von Typen bevorzugt. Mit Hilfe von Erweiterungselementen können aber auch andere Typ-Systeme eingebunden werden. Statt die Typen im WSDL-Dokument zu definieren, kann man auch bereits existierende XML-Schema-Dokumente einbinden (mit Hilfe von import) und die darin definierten Datentypen verwenden.

Nachrichten

Nachrichten dienen in WSDL dazu, das Format der Ein- und Ausgabedaten von Diensten festzulegen. Jede Nachricht wird in einem message-Element definiert. Die Syntax dazu sieht folgendermaßen aus:

```
<definitions …>
  <message name="nmtoken"> *
    <part name="nmtoken" element="qname"? type="qname"? /> *
  </message>
</definitions>
```

Ein WSDL-Dokument kann beliebig viele Nachrichten (und damit message-Elemente) enthalten. Jede Nachricht besteht aus einem oder mehreren logischen Bestandteilen, die jeweils in einem part-Element definiert werden. Wenn man Nachrichten mit Funktionen vergleicht, dann entsprechen die Bestandteile einer Nachricht den Funktionsparametern. Der Name einer Nachricht (Attribut name) muss eindeutig innerhalb aller Nachrichten des Dokuments sein. Der Name eines Bestandteils muss eindeutig innerhalb aller Bestandteile der umschließenden Nachricht sein. Jedem Bestandteil wird mit Hilfe von speziellen Attributen ein Typ (aus einem Typsystem) zugeordnet. Für die Verwendung von XML-Schema-Typen bietet WSDL standardmäßig die Attribute element und type an. Die Typbeschreibung einer Nachricht gibt deren Inhalt nur abstrakt wieder, d. h. das tatsächliche Format zur Übertragung der Nachricht kann – abhängig von der verwendeten Bindung – durchaus davon abweichen.

Im WSDL-Dokument 10.1 werden die beiden Nachrichten temperaturAbfrageSoapIn und temperaturAbfrageSoapOut definiert. Beide bestehen aus je einem Bestandteil. Während der Bestandteil datum der Nachricht temperaturAbfrageSoapIn mit dateTime einen Typ besitzt, der bereits in XML-Schema definiert ist, ist der Typ des Bestandteils Result benutzerdefiniert und stammt aus dem types-Element.

Port-Typen

WSDL verwendet Port-Typen, um eine Menge von abstrakten

Operationen und die dazugehörigen abstrakten Nachrichten unter einem gemeinsamen Namen zusammenzufassen. Die Syntax dazu sieht folgendermaßen aus:

```
<definitions …>
  <portType name="nmtoken"> *
    <operation name="nmtoken" … /> *
  </portType>
</definitions>
```

Der Name eines Port-Typs muss eindeutig innerhalb aller Port-Typen eines WSDL-Dokuments sein. Jede Operation wird in einem eigenen operation-Element definiert. Der Name einer Operation muss eindeutig innerhalb des umschließenden Port-Typs sein. Grundsätzlich stehen vier Operationsprimitiven zur Verfügung, die definieren welche Nachrichten und wie Nachrichten empfangen und gesendet werden:

- One-Way: Der Endpunkt empfängt eine Nachricht
- Request-Response: Der Endpunkt empfängt eine Nachricht und sendet eine korrelierte Nachricht als Antwort.
- Solicit-Response: Der Endpunkt sendet eine Nachricht und empfängt eine korrelierte Nachricht als Antwort.
- Notification: Der Endpunkt sendet eine Nachricht.

Ein input-Unterelement eines operation-Elements definiert das abstrakte Nachrichtenformat für eine zu empfangende Nachricht. Ein output-Unterelement definiert das Format der zu sendenden Nachricht. Ein fault-Unterelement definiert mögliche auftretende Fehlernachrichten. Art und Reihenfolge der Unterelemente eines operation-Elements bestimmen den Typ einer Operation. Operationen vom Typ Request-Response besitzen ein input, ein output- und ein oder mehrere fault-Unterelemente (in genau dieser Reihenfolge). Solicit-Response-Operationen können ebenfalls diese unterschiedlichen Unterelemente haben, allerdings in der Reihenfolge output-Unterelement, input-Unterelement und fault-Unterelemente. Request-Response- und Solicit-Response-Operationen unterscheiden sich also nur in der Reihenfolge, in der ihre Unterelemente definiert werden. One-Way-Operationen besitzen nur ein einzelnes input-Unterelement, Notification-Operationen nur ein einzelnes output-Unterelement. Jedem der Unterelemente wird mit Hilfe eines message-Attributs eine Nachricht zugeordnet. Der Name jedes Unterelements muss eindeutig innerhalb des umschließenden Port-Typs sein. Wird einem der Elemente kein Name zugewiesen, ordnet WSDL automatisch einen Namen zu. Dieser entspricht dem Namen der umschließenden Operation im Falle von One-Way- oder Notification-Operationen. Im Falle von Request-Response- und Solicit-Response-Operationen entspricht der Name dem Namen der Operation mit einem angehängten

„Request“ bzw. „Solicit“ für die Eingangsnachricht und einem angehängten „Response“ für die Ausgangsnachricht.

Folgender Auszug aus dem WSDL-Dokument 10.1 definiert die Operation temperaturAbfrage:

```
<operation name="temperaturAbfrage" parameterOrder="datum">
   <input name="temperaturAbfrageSoapIn" message="tns:temperaturAbfrageSoapIn"/>
   <output name="temperaturAbfrageSoapOut" message="tns:temperaturAbfrageSoapOut"/>
</operation>
```

Diese Operation ist vom Typ Request-Response, da zuerst ein input-Element und dann ein output-Element definiert wird (fault-Elemente wurden keine definiert). Die Dokumente SOAP-Dokument 10.2 und SOAP-Dokument 10.3, die bereits in Abschnitt 10.3 gezeigt wurden, sind Beispiele für Ein- und Ausgabenachrichten dieser Operation. Wird eine Nachricht später zusammen mit einer RPC-Bindung verwendet, besteht bei Request-Response- und Solicit-Response-Operationen die Möglichkeit, die Reihenfolge der Parameter anzugeben, wie sie in der Signatur der ursprünglichen RPC-Funktion festgelegt ist. Dazu dient das Attribut parameterOrder. Der Wert dieses Attributs ist im Allgemeinen eine Liste von Namen von Nachrichtenbestandteilen, jeweils durch ein Leerzeichen getrennt.

10.5.2. Verknüpfung mit Kommunikationsprotokollen und Nachrichtenformaten

Mit den bisherigen Definitionen wurden Typen, abstrakte Nachrichten, Operationen und Port-Typen definiert. Die nun folgenden Definitionen für Bindungen, Ports und Dienste binden diese abstrakten Definitionen an tatsächliche Nachrichtenformate, Protokolle und Netzwerkadressen.

Bindungen

Eine Bindung legt für einen bestimmten Port-Typ ein spezifisches Protokoll und Nachrichtenformat für die Übertragung fest. Einem Port-Typ können mehrere unterschiedliche Bindungen zugeordnet werden, z. B. Bindungen für SOAP, für HTTP und für MIME. Die Syntax einer Bindung sieht folgendermaßen aus:

```
<definitions .... >
  <binding name="nmtoken" type="qname"> *
    <!-- Erweiterungselemente -->
    <operation name="nmtoken"> *
      <!-- Erweiterungselemente -->
      <input name="nmtoken"? > ?
        <!-- Erweiterungselemente -->
      </input>
      <output name="nmtoken"? > ?
        <!-- Erweiterungselemente -->
      </output>
      <fault name="nmtoken"> *
        <!-- Erweiterungselemente -->
      </fault>
    </operation>
  </binding>
</definitions>
```

Zur Definition einer Bindung wird das binding-Element verwendet. Der Name der Bindung muss eindeutig innerhalb aller Bindungen eines WSDL-Dokuments sein. Das type-Attribut legt den Port-Typ fest, für den die Bindung definiert wird. Die tatsächlichen Bindungsinformationen werden mit Hilfe von Erweiterungselementen definiert, da die Art der Informationen und damit die Menge (Namen und Anzahl) der Elemente von der Art der Bindung abhängig sind. WSDL ermöglicht die Definition von Bindungsinformationen mit Hilfe von Erweiterungselementen für zu empfangende Nachrichten (im input-Element), zu sendende Nachrichten (im output-Element), und Fehlernachrichten (in den fault-Elementen). Außerdem können Erweiterungselemente benutzt werden, um Bindungsinformationen für spezifische Operationen (in den operation-Elementen) sowie ganz allgemein für die Bindung (im binding-Element) anzugeben.

Das WSDL-Dokument 10.1 enthält eine Bindung für das SOAP-Protokoll mit Namen TemperaturdienstSoap. Protokollspezifische Informationen werden mit Hilfe der Erweiterungselemente soap:binding, soap:operation und soap:body festgelegt.

Ports und Dienste

Ein Port definiert einen individuellen Endpunkt im Netz, indem er einer Bindung eine einzelne Netzwerkadresse zuordnet. Ein Dienst gruppiert schließlich mehrere zusammengehörige Ports. Ports werden in einem port-Element definiert, Dienste in einem service-Element. Die Syntax dazu sieht folgendermaßen aus:

```
<definitions … >
  <service name="nmtoken"> *
    <port name="nmtoken" binding="qname"> *
      <!-- Erweiterungselemente -->
    </port>
  </service>
</definitions>
```

Der Name eines Ports muss eindeutig innerhalb aller Ports eines WSDL-Dokuments sein. Der Name eines Dienstes muss eindeutig innerhalb aller Dienste eines WSDL-Dokuments sein. Das binding-Attribut eines Ports legt die Bindung fest, für die ein Port definiert wird. Die Zuordnung der Netzwerkadresse geschieht mit Hilfe von Erweiterungselementen. Das WSDL-Dokument 10.1 beispielsweise enthält eine Bindung an SOAP. Mit Hilfe des Erweiterungselements soap:address wird die SOAP-Adresse des Temperaturdienstes festgelegt, im Beispiel auf die URL

`http://wetter.fmi.uni-passau.de:8004/service/Temperaturdienst`.

Neben der Bindung an das SOAP-Protokoll, die in dem Beispieldokument verwendet wird, beschreibt die Spezifikation von WSDL außerdem noch Bindungen an das HTTP-Protokoll und an das MIME-Protokoll.

Ports, die zu einem Dienst zusammengefasst werden, sollten nicht miteinander kommunizieren. Außerdem geht WSDL davon aus, dass Ports, die auf denselben Port-Typ verweisen, als Alternativen anzusehen sind, d. h. dass ihre Funktionsweise semantisch gesehen gleich ist.

10.5.3. Einbettung von WSDL in UDDI

Ein WSDL-Dokument dient dazu, Schnittstellen und Bindungen an Protokolle und Datenformate für Dienste festzulegen. Diese Informationen werden zum Teil auch für die Beschreibung eines Dienstes in UDDI benötigt. Die entsprechenden Standardisierungsgremien haben sich deshalb auf einen Vorschlag geeinigt, um WSDL-Dokumente in UDDI zu verwenden [CuER01].

Dabei gehen die Standardisierungsgremien davon aus, dass entsprechende Gremien aus Industrie, Forschung und Politik eine Menge von Diensttypen festlegen und diese mit Hilfe von WSDL-Dokumenten beschreiben. Diese Dokumente enthalten die Beschreibung der Schnittstellen der Dienste sowie Bindungen für verschiedene Protokolle, nicht aber Port- und Dienst-Definitionen. Die WSDL-Dokumente werden als UDDI-tModels registriert. Das overviewDoc-Element eines solchen tModels verweist dabei auf das entsprechende WSDL-Dokument (mit

Hilfe einer URL). Dienste, die das entsprechende tModel implementieren, müssen dann eine Schnittstelle aufweisen, wie sie im WSDL-Dokument festgelegt ist.

Wie bereits erwähnt, enthalten derartige WSDL-Dokumente keine Port- und Dienst-Definitionen. Diese Informationen werden in den entsprechenden UDDI-Datenstrukturen beschrieben. Zum Beispiel wird die tatsächliche Netzwerkadresse eines Dienstes im accessPoint-Element des dazugehörigen bindingTemplates beschrieben.

10.6. Dienstkomposition und -interaktion

In Abschnitt 10.2 haben wir schon kurz die Dienste Einheitenumrechnerdienst und TemperaturInFahrenheitDienst vorgestellt. TemperaturInFahrenheitDienst liefert dabei, genauso wie der Temperaturdienst, die Temperatur zu einem gegebenen Zeitpunkt zurück. Allerdings, und das ist der einzige wirkliche Unterschied, liefert TemperaturInFahrenheitDienst sein Ergebnis im Gegensatz zum Temperaturdienst immer in Grad Fahrenheit. Es ist also sinnvoll, den Dienst TemperaturInFahrenheitDienst nicht von Grund auf neu zu implementieren, sondern auf die vorhandenen Dienste Temperaturdienst und Einheitenumrechnerdienst zurückzugreifen. Abbildung 10.7 zeigt, wie der TemperaturInFahrenheitDienst arbeitet. Der zusammengesetzte Dienst ruft, sobald er eine Anfrage erhält, den Temperaturdienst auf, um die Temperatur in Grad Celsius zu dem angegebenen Zeitpunkt zu ermitteln. Das Ergebnis des Temperaturdienstes muss er dann in geeigneter Form an den Einheitenumrechnerdienst schicken, damit dieser die Grad Celsius in Grad Fahrenheit umrechnet. Abschließend muss der TemperaturInFahrenheitDienst die Antwort auf die Anfrage im passenden Format zurückschicken.

Der Dienst TemperaturInFahrenheitDienst hat damit im Grunde nur zwei Aufgaben zu erfüllen. Zum einen muss er die beiden Basisdienste ansprechen und ihnen dabei die richtigen Parameter zur Verfügung stellen. Zum anderen muss er das Ergebnis des Temperaturdienstes auswerten, daraus die notwendigen Parameter für den Einheitenumrechnerdienst extrahieren und das Ergebnis des Einheitenumrechnerdienstes auswerten, um daraus sein eigenes Ergebnis zu ermitteln.

Um derartige Kompositionen von Web Services automatisieren zu können, wurden Sprachen basierend auf XML entwickelt. Diese erlauben die Spezifikation, welche Dienste wann angesprochen werden sollen und

wie die Ergebnisdokumente eines Dienstes transformiert werden müssen, um als Eingabe für den nächsten Dienst geeignet zu sein. Beispiele für solche Sprachen sind die Web Services Flow Language (WSFL) [Leym01] von IBM, XL [FlKo01] und XLang [That01] von Microsoft. Diese Sprachen unterstützen nicht nur die sequentielle Verkettung von Diensten wie im obigen (einfachen) Beispiel sondern erlauben die Spezifikation eines komplexen Ablaufs, ähnlich wie dies etwa bei Workflows möglich ist. Welche Möglichkeiten im Einzelnen bestehen, um Dienste zu kombinieren, hängt von der verwendeten Sprache ab.

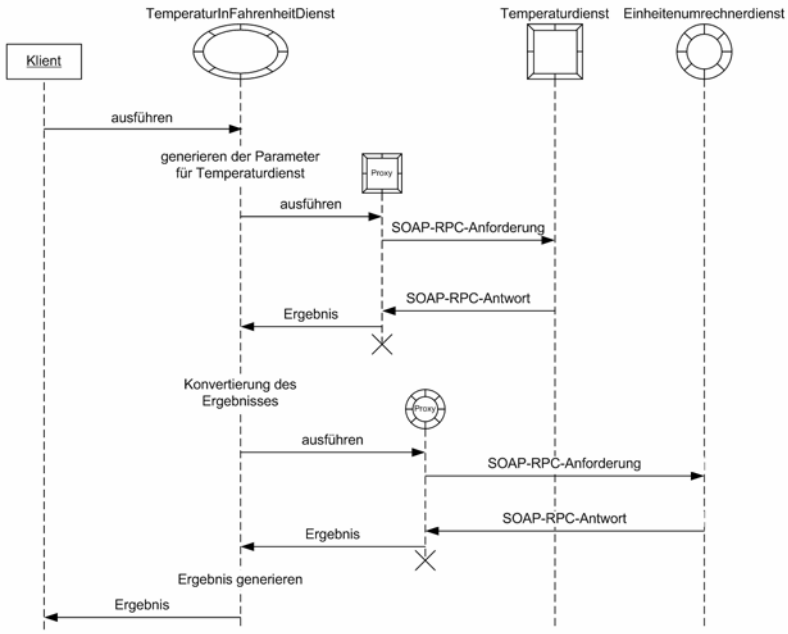


Abbildung 10.7: Dienstkomposition am Beispiel TemperaturInFahrenheitDienst

Der TemperaturInFahrenheitDienst kann natürlich öffentlich zugänglich gemacht werden, analog zu den Diensten Temperaturdienst und Einheitenumrechnerdienst. Dazu ist es zum einen notwendig, ein passendes WSDL-Dokument zu erstellen. Dieses unterscheidet sich vom WSDL-Dokument des Temperaturdienstes, da eine Rückgabe der Einheit der Temperatur unnötig ist, weil die Einheit auf Grad Fahrenheit festgelegt ist. Zum anderen müssen die entsprechenden Einträge in UDDI eingefügt werden. Diese bestehen aus einem entsprechenden tModel, das auf das neue WSDL-Dokument für den Dienst TemperaturInFahrenheitDienst verweist, und geeigneten bindingTemplate- und businessService-Einträgen, falls diese noch nicht

existieren. Alternativ hätte man auch das tModel des Temperaturdienstes verwenden können, allerdings müsste man in diesem Fall im Ergebnis des TemperaturInFahrenheitDienstes die Einheit angeben.

10.7. Plattformen, Produkte und Infrastrukturen

Aufgrund des Potentials von Web Services das Internet in eine Dienstplattform zu verwandeln, wird ihnen von Industrie und Forschung immer größere Aufmerksamkeit geschenkt. Produkte und Plattformen wie HP Web Services Platform, Sun ONE und Microsoft .NET zeigen dies. Die Produkte und Plattformen basieren auf den in den vorhergehenden Abschnitten vorgestellten Standards und Technologien, wie SOAP, UDDI und WSDL. Sie stellen Werkzeuge zur Verfügung, mit denen z. B. existierende Anwendungen unkompliziert und schnell als Dienste zur Verfügung gestellt werden können, einschließlich der Generierung von entsprechenden WSDL-Dokumenten, passenden UDDI-Einträgen, usw. Außerdem unterstützen sie die Komposition und Interaktion dieser Dienste und ermöglichen damit die Interaktion der dahinter stehenden Anwendungen. Auf diese Art und Weise ist es möglich, Business-to-Business-Integration zu realisieren. Dabei entwickelt man für vorhandene Anwendungen eine Web-Service-Schnittstelle, über die dann beispielsweise die ERP-Systeme verschiedener Firmen interagieren können.

Neben diesen kommerziellen Produkten und Plattformen existieren Forschungsprototypen wie etwa ServiceGlobe [KSSK02] und SELF-SERV [BDSN02]. Das ServiceGlobe-System soll im Folgenden als eine Beispielplattform beschrieben werden.

Das ServiceGlobe-System stellt eine neuartige Infrastruktur für die Implementierung und Ausführung von Web Services zur Verfügung. Es ermöglicht die Verwaltung von und die Suche nach Web Services, basierend auf den Standards UDDI und WSDL. Web Services können auf beliebigen Rechnern im Internet, die in die ServiceGlobe-Föderation integriert sind, verteilt ausgeführt und dynamisch aufgerufen werden. ServiceGlobe stellt unter anderem Standardfunktionalität von Dienstplattformen zur Verfügung, z. B. Kommunikation auf SOAP/XML-Basis, Transaktions- und Sicherheitssystem. Zusätzlich werden auch verschiedene Optimierungen wie etwa Lastbalancierung unterstützt.

ServiceGlobe unterscheidet grundsätzlich zwischen zwei Arten von Diensten: externe und interne Dienste. Externe Dienste sind existierende,

stationäre Dienste, wie sie zurzeit im Internet zu finden sind, die nicht von ServiceGlobe selbst angeboten werden. Derartige Dienste können auf beliebigen Systemen im Internet implementiert sein und beliebige Schnittstellen zum Aufruf anbieten. ServiceGlobe ermöglicht die Verwendung solcher Dienste unabhängig von ihrer tatsächlichen Schnittstelle, z. B. SOAP oder RPC, mit Hilfe von Adaptoren. Diese kapseln die Kommunikation mit externen Diensten und bieten damit eine einheitliche Schnittstelle an. Damit erfüllen sie ähnliche Aufgaben wie Wrapper in Mediatorensystemen.

Interne Dienste sind in ServiceGlobe ausgeführte Dienste. Sie sind in Java implementiert, basierend auf der API, die von ServiceGlobe zur Verfügung gestellt wird. ServiceGlobe verwendet XML zur Kommunikation mit anderen Diensten, d. h., ein Dienst empfängt ein einzelnes XML-Dokument als Eingabe und er generiert ein einzelnes XML-Dokument als Ausgabe. Es existieren zwei Arten von internen Diensten: dynamische und statische Dienste. Statische Dienste sind ortsabhängig, d. h., sie können nicht dynamisch auf beliebigen ServiceGlobe-Servern ausgeführt werden. Derartige Dienste benötigen möglicherweise Zugriff auf bestimmte lokale Ressourcen, z. B. ein lokales DBMS, um den internen Zustand abzuspeichern, oder sie benötigen bestimmte Rechte, z. B. für den Zugriff auf das Dateisystem, die nur auf bestimmten Servern verfügbar sind. Diese Einschränkungen verhindern eine Ausführung von statischen Diensten auf beliebigen ServiceGlobe-Servern. Ein Beispiel für einen statischen Dienst ist der Temperaturdienst, der Zugriff auf einen Temperatursensor benötigt und damit nicht auf beliebigen Servern ausgeführt werden kann. Im Gegensatz dazu sind dynamische Dienste ortsunabhängig. Sie sind zustandslos, d. h., der interne Zustand derartiger Dienste wird nach der Abarbeitung einer Anfrage verworfen, und sie benötigen keine speziellen Ressourcen oder Rechte. Deshalb können sie auf jedem beliebigen ServiceGlobe-Server ausgeführt werden. Beispiele für dynamische Dienste sind der Einheitenumrechnerdienst und der TemperaturInFahrenheitDienst.

Es gibt eine orthogonale Klassifikation von internen Diensten in Adaptoren, einfache Dienste und zusammengesetzte Dienste. Adaptoren wurden bereits eingeführt. Einfache Dienste sind interne Dienste, die keinen anderen Dienst verwenden. Der Einheitenumrechnerdienst ist zum Beispiel ein einfacher Dienst. Zusammengesetzte Dienste verwenden andere Dienste, die in diesem Zusammenhang auch als Basisdienste bezeichnet werden. Ein Beispiel für einen zusammengesetzten Dienst ist der TemperaturInFahrenheitDienst.

Natürlich kann ein zusammengesetzter Dienst auch selbst wieder von einem weiteren zusammengesetzten Dienst als Basisdienst verwendet werden.

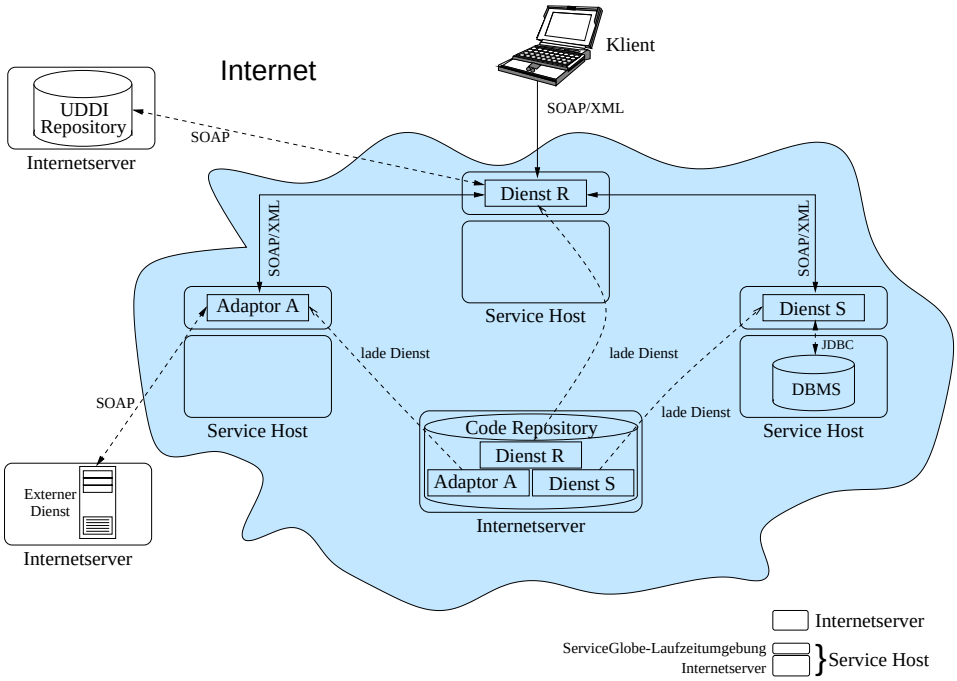


Abbildung 10.8: Architektur des ServiceGlobe-Systems

Interne Dienste werden auf so genannten Service Hosts ausgeführt. Dabei handelt es sich um gewöhnliche Internetserver, auf denen zusätzlich eine ServiceGlobe-Laufzeitumgebung läuft. Sie kommunizieren mit anderen Diensten mittels SOAP-Nachrichten. Die Kommunikation mit externen Diensten wird durch Adaptoren gekapselt, welche Anfragen auf die Schnittstelle des externen Dienstes umsetzen. Interne Dienste sind mobiler Code. Wenn ihr ausführbarer Code benötigt wird, wird er auf Anforderung von so genannten Code Repositories auf Service Hosts geladen. UDDI wird verwendet, um ein passendes Code Repository für einen bestimmten Dienst zu finden.

Abbildung 10.8 gibt einen Überblick über die wesentlichen Komponenten des ServiceGlobe-Systems. Zu Beginn sendet der Klient (mit Hilfe von SOAP) einen Aufruf an einen Service Host, den dynamischen Dienst R auszuführen. Wenn der ausführbare Code dieses Dienstes noch nicht lokal gepuffert ist, wird er von einem Code Repository geladen und auf dem Service Host instanziiert. Während seiner Ausführung greift Dienst R auf die Basisdienste Adaptor A und

Dienst S zu. Dazu werden die beiden Dienste zuerst mit Hilfe von UDDI gesucht. Wenn nötig, wird Adaptor A auf einen passenden Service Host geladen und dort im Auftrag von Dienst R ausgeführt. Dienst S ist ein statischer Dienst. Er läuft bereits und kann mit Hilfe von SOAP-Nachrichten angesprochen werden. Im Folgenden wird nun die Ausführung eines internen Dienstes in ServiceGlobe beschrieben, die sich grob in vier Schritte aufteilen lässt:

Dienstspezifikation: Unter Dienstspezifikation versteht man den Prozess der Programmierung eines Dienstes. Eine Möglichkeit dazu ist die Verwendung von Java und der API von ServiceGlobe. Komfortabler ist es allerdings, eine spezialisierte Programmiersprache, z. B. XL [FlKo01], oder ein Werkzeug zur grafischen Dienstkomposition (ähnlich wie bei Workflows) zu verwenden. Damit man in ServiceGlobe ausführbare Dienste erhält, muss sowohl ein Programm als auch eine grafische Repräsentation in eine oder mehrere Java-Klassen kompiliert werden, welche die ServiceGlobe-API verwenden.

Dynamische Dienstauswahl: UDDI ordnet jedem Dienst ein tModel zu, das im Grunde die Semantik und die Schnittstellen des Dienstes beschreibt. Ein Dienst ist somit eine Implementierung seines tModels. In ServiceGlobe müssen zusammengesetzte Dienste nicht unbedingt einen konkreten Dienst aufrufen. Es reicht, wenn sie das tModel angeben oder „aufrufen“. Eine passende Implementierung wird dann während der Kompilierung oder zur Laufzeit ausgewählt. Dieser Vorgang der Auswahl eines Dienstes für ein tModel wird dynamische Dienstauswahl genannt. Dynamische Dienstauswahl ist nicht darauf beschränkt, nur eine Implementierung für ein tModel auszuwählen. Da UDDI bei einer Anfrage alle Dienste zurückliefert, die ein gegebenes tModel implementieren, können mehrere Implementierungen ausgewählt und folglich auch mehr als ein Dienst aufgerufen werden. Dadurch ist es zum Beispiel möglich, auf die schnellste Antwort zu warten.

Dienstverteilung: Der Hauptvorteil von dynamischen Diensten ist ihre Ortsunabhängigkeit. Zur Laufzeit ermöglicht dies die dynamische Verteilung solcher Dienste auf beliebige Service Hosts, wodurch sich eine Vielzahl an Optimierungsmöglichkeiten ergibt. Mehrere Instanzen eines dynamischen Dienstes können auf verschiedenen Rechnern ausgeführt werden, um Lastbalancierung und Parallelisierung zu erreichen. Dynamische Dienste können auf den Service Hosts ausgeführt werden, die optimale Voraussetzungen aufweisen, z. B. einen schnellen Prozessor, ausreichend Hauptspeicher oder eine schnelle Netzwerkverbindung. Zusammen mit dem Laden von Diensten zur Laufzeit ergibt sich eine große Flexibilität im Hinblick auf

Lastbalancierung und Optimierung. ServiceGlobe besitzt ein umfassendes Sicherheitssystem, das die Sicherheitsaspekte von mobilem Code berücksichtigt und somit Service Hosts vor bösartigen Diensten schützt. Ausführliche Beschreibungen von Sicherheitssystemen, die dem von ServiceGlobe gewählten Ansatz ähnlich sind, finden sich in [BKKK01, SeBK01, BrKK99] und [KrIK00, KSKK99].

Dienstladen zur Laufzeit: Nach ihrer Verteilung müssen dynamische Dienste von Code Repositories geladen und auf den ausgewählten Service Hosts ausgeführt werden. Dadurch ist die Menge der verfügbaren Dienste nicht fest, sondern sie kann zur Laufzeit durch jeden Teilnehmer der ServiceGlobe-Föderation erweitert werden.

10.8. Zusammenfassung und Ausblick

In diesem Kapitel haben wir die wichtigsten Standards und Technologien im Bereich Web Services vorgestellt. Web Services verwandeln das Internet immer mehr zu einer Plattform, auf der Dienste angeboten werden, und sie ermöglichen vollautomatische Dienstnutzung und Dienstkomposition. Anhand eines einfachen Beispiel-Web-Dienstes, einem Temperaturdienst für die Stadt Passau, haben wir die wichtigsten Standards im Bereich Web Services für Datenaustausch, Dienstverwaltung und Dienstbeschreibung erklärt: SOAP, UDDI, WS-Inspection und WSDL.

SOAP ist ein Kommunikationsprotokoll für verteilte Anwendungen, das es ermöglicht, strukturierte und typisierte Daten mit Hilfe von XML auszutauschen. UDDI ist ein globales Verzeichnis für Dienst-Metadaten, das die Speicherung von Dienst-Metadaten in einer einheitlichen Datenstruktur (UDDI-Schema) an zentralen und öffentlich zugänglichen Stellen im Internet (UDDI-Server) durch einheitliche Veröffentlichungs-Mechanismen ermöglicht. Außerdem unterstützt UDDI die Metadatenabfrage durch eine normierte Anfragesprache. Der WS-Inspection-Standard definiert, wie Dienstanbieter Metadaten ihrer Dienste auch – alternativ oder zusätzlich zu der Speicherung in einem UDDI-Verzeichnis – in standardisierter Form auf ihrem Web-Server anbieten können, indem sie dort entsprechende WS-Inspection-Dokumente ablegen. WSDL schließlich legt fest, auf welche Weise man die zum Aufruf eines Dienstes notwendigen Informationen in einem XML-Dokument spezifiziert. Zu diesen Informationen zählen z. B. die Operationen, die der Dienst unterstützt, das Format der Eingabe- und Ausgabedaten und das Protokoll, das zur Kommunikation mit dem

Dienst verwendet werden muss.

Neben diesen Basistechnologien sind wir auch darauf eingegangen, wie Web Services die Komposition von neuen, zusammengesetzten Diensten unter Verwendung von bestehenden Diensten erleichtern. Wir haben anhand des Beispieldienstes TemperaturInFahrenheitDienst gezeigt, wie einfach es ist einen neuen Dienst aus bestehenden Diensten zusammenzubauen. Neuartige, spezialisierte Sprachen zur Dienstkomposition, wie z. B. WSFL, XL und XLang, werden dies in Zukunft noch einfacher machen. Abschließend haben wir als eine Beispielplattform das ServiceGlobe-System beschrieben.

Noch ist die Entwicklung im Bereich Web Services im Anfangsstadium. Das kann man unter anderem daran erkennen, dass selbst die grundlegenden Standards SOAP, UDDI und WSDL noch Änderungen unterworfen sind. Sprachen zur Dienstkomposition sind gerade erst im Entstehen begriffen. Gleiches gilt für Standards, die das Schema der Ein- und Ausgabedaten für bestimmte Dienste, z. B. im Bereich B2B, festlegen. Ein Beispiel hierfür ist die Universal Business Language (UBL).[11] Das zuständige Komitee hat sich zum Ziel gesetzt, eine Standardbibliothek für XML-Dokumente aus dem Geschäftsbereich zu definieren, z. B. für Bestellungen und Rechnungen. Doch trotz dieser Probleme zeigt gerade das Engagement vieler bedeutender IT-Firmen, dass Web Services sich durchsetzen werden.

Literatur

ABDK02 Atkinson, B.; Brown, A.; Della-Libera, G.; Kaler, C.; Klein, J.; Konersmann, S.; Leach, P.; Manferdelli, J.; Shewchuk, J.; Simon, D.: Web Services Security Language (WS-Security). http://msdn.microsoft.com, 2002.

BBMN01 Ballinger, K.; Brittenham, P.; Malhotra, A.; Nagy, W. A.; Pharies, S.: Web Services Inspection Language (WS-Inspection) 1.0. http://www.ibm.com/developerworks/webservices/library/ws-wsilspec.html, 2001.

BDSN02 Benatallah, B.; Dumas, M.; Sheng, Q. Z.; Ngu, A. H. H.: Declarative Composition and Peer-to-Peer Provisioning of

[11] Siehe `http://www.oasis-open.org/committees/ubl/`

Dynamic Web Services. In: Proceedings of the 18th International Conference on Data Engineering IEEE Computer Society (ICDE), 2002, S. 297-308.

BEKL00 Box, D.; Ehnebuske, D.; Kakivaya, G.; Layman, A.; Mendelsohn, N.; Nielsen, H.F.; Thatte, S.; Winer, D.: Simple Object Access Protocol (SOAP) 1.1. http://www.w3c.org/TR/SOAP/, 2000.

BKKK01 Braumandl, R.; Keidl, M.; Kemper, A.; Kossmann, D.; Kreutz, A.; Seltzsam, S.; Stocker, K.: ObjectGlobe: Ubiquitous Query Processing on the Internet. In: The VLDB Journal: Special Issue on E-Services, Volume 10, Nummer 3, 2001, S. 48-71.

BrKK99 Braumandl, R.; Kemper, A.; Kossmann, D.: Database Patchwork on the Internet: Project Demo. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1999, S. 550-552.

CCMW01 Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S.: Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl/, 2001.

CuER01 Curbera, F.; Ehnebuske, D.; Rogers, D.: Using WSDL in a UDDI Registry. http://www.uddi.org/bestpractices.html, 2001.

Dada96 Dadam, P.: Verteilte Datenbanken und Client/Server-Systeme. Springer-Verlag, Berlin, 1996.

FlKo01 Florescu, D.; Kossmann, D.: An XML Programming Language for Web Service Specification and Composition. In: IEEE Data Engineering Bulletin, Volume 24, Nummer 2, 2001, S. 48-56.

KeEi01 Kemper, A.; Eickler, A.: Datenbanksysteme. Eine Einführung. Oldenbourg Verlag, 4. erweiterte Auflage, 2001.

KeWi01 Kemper, A.; Wiesner, C.: HyperQueries: Dynamic Distributed Query Processing on the Internet. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), 2001, S. 551-560.

KKKK01 Keidl, M.; Kreutz, A.; Kemper, A.; Kossmann, D.: Verteilte Metadatenverwaltung für die Anfragebearbeitung

auf Internet-Datenquellen. In: Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 9. GI-Fachtagung, 2001, S. 107-126.

KKKK02 Keidl, M.; Kreutz, A.; Kemper, A.; Kossmann, D.: A Publish & Subscribe Architecture for Distributed Metadata Management. In: Proceedings of the 18th International Conference on Data Engineering IEEE Computer Society (ICDE), 2002, S. 309-320.

KrIK00 Krivokapic, N.; Islinger, M.; Kemper, A.: Migrating Autonomous Objects in a WAN Environment. In: Journal of Intelligent Information Systems, Volume 15, Nummer 3, 2000, S. 221-252.

KSKK99 Keidl, M.; Seltzsam, S.; Kemper, A.; Krivokapic, N.: Sicherheit in einem Java-basierten verteilten System autonomer Objekte. In: Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 8. GI-Fachtagung, 1999, S. 38-58.

KSSK02 Keidl, M.; Seltzsam, S.; Stocker, K.; Kemper, A.: ServiceGlobe: Distributing E-Services across the Internet. Projektvorführung auf der 28th International Conference on Very Large Data Bases (VLDB), 2002.

KSWD02 Kraiß, A.; Schön, F.; Weikum, G.; Deppisch, U.: Mit HEART zu Middleware-basierten Antwortzeitgarantien für E-Services. In: Datenbank-Spektrum, Band 20, Heft 1, 2002, S. 64-74.

Leym01 Leymann, F.: Web Services Flow Language (WSFL 1.0). http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, 2001.

ScSt00 Scribner K.; Stiver M.C.: Understanding SOAP. Sams Publishing, 2000.

SeBK01 Seltzsam, S.; Börzsönyi, S.; Kemper, A.: Security for Distributed E-Service Composition. In: Proceedings of the 2nd International Workshop on Technologies for E-Services (TES), 2001, S. 147-162.

That01 Thatte, S.: XLANG - Web Services for Business Process Design. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/, 2001.

UDDI Universal Description, Discovery and Integration (UDDI). http://www.uddi.org.

WiWK02 Wiesner, C.; Winklhofer, P.; Kemper, A.: Building Dynamic Market Places Using HyperQueries. Projektvorführung auf der VIII. Conference on Extending Database Technology (EDBT), 2002.

10.9. Glossar

Dienst	siehe *Web Service*
Dienstkomposition	Entwicklung von *zusammengesetzten Diensten* mit Hilfe von anderen *Diensten*
Dienst-Metadaten	Summe der Daten über Dienstanbieter, Dienste und technische Spezifikationen
E-Business	Abwicklung von Geschäftsabläufen über das Internet oder andere Computernetzwerke
ebXML	Modulare Zusammenstellung von Standards im Bereich *E-Business*
Electronic-Business	siehe *E-Business*
FTP	Protokoll, das verwendet wird, um Dateien zwischen Rechnern zu übertragen
HTTPS	Kombination des *HTTP*-Protokolls mit *SSL* um eine verschlüsselte Übertragung von Daten zu ermöglichen
MIME	Ein Standardformat für Email. Mit Hilfe verschiedener *MIME*-Typen ist es möglich Audio-, Video-, Bild- oder HTML-Dateien zu versenden.
Proxy	Übernimmt die Kommunikation zwischen einem Benutzer und einem *Web Service* (im Kontext *Web Service*)
RPC	Aufruf von Funktionen über Rechnergrenzen hinweg.
Service	siehe *Web Service*
ServiceGlobe	Dienstplattform und System für verteilte und erweiterbare *Dienstkomposition*

SMTP	Protokoll, das verwendet wird, um Nachrichten (Emails) an andere Rechner zu verschicken.
SOAP	Ein auf XML basierendes Protokoll für den Datenaustausch, das unter anderem *RPC*-Aufrufe ermöglicht
SSL	Verfahren zur sicheren Übertragung von Daten, das in Kombination mit *HTTP*, *FTP*, und anderen Kommunikationsprotokollen einsetzbar ist.
UDDI	Verzeichnisdienst und –standard für *Dienst*-Metadaten
UDDI-Wolke	Serververbund zur Realisierung eines globalen *UDDI*-Verzeichnisdienstes
UUID	Weltweit eindeutiger Identifikator für *Dienst*-Metadaten-Dokumente (im Kontext *UDDI*)
Web Service	Bisher keine einheitliche Definition des Begriffs. Im üblichen Sprachgebrauch bezeichnet ein Web Service einen Dienst, der Benutzern über das Web zur Verfügung gestellt wird und dabei beispielsweise auf XML und HTTP zurückgreift und auf automatisierte Benutzung ausgerichtet ist.
Web Service Engine	Anwendung, die es erlaubt, *Web Services* im Internet verfügbar zu machen
WSDL	Standard zur Beschreibung der Schnittstellen von *Web Services*
WSFL	Standard zur Beschreibung von *zusammengesetzten Diensten*
WS-Inspection	Standard zur Bereitstellung von Verweisen auf Informationen über *Web Services* auf einem *Web-Server*
XL	Sprache zur Beschreibung von *Diensten* und *zusammengesetzten Diensten*
XLANG	Sprache zur Beschreibung von *zusammengesetzten Diensten*

Zusammengesetzte Dienste	*Dienst*, der andere *Dienste* anspricht und ihre Ergebnisse verwendet (siehe *Dienstkomposition*)

10.10. Abkürzungsverzeichnis

ebXML	Electronic Business using eXtensible Markup Language
FTP	File Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
MIME	Multipurpose Internet Mail Extensions
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
UDDI	Universal Description, Discovery and Integration
UUID	Universally Unique Identifier
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WS-Inspection	Web Services Inspection Language