Fachhochschule
Bonn-Rhein-Sieg

University
of Applied Sciences

Seminararbeit im Fach „verteilte und parallele Systeme I“
6. Fachsemester
SS2002

zum Thema

UDDI

(Universal Description Discovery and Integration)

von

Reto Kortas
Christoph Ersfeld

11. Juni 2002

Inhaltsverzeichnis

2 Gründe für die Entwicklung von UDDI

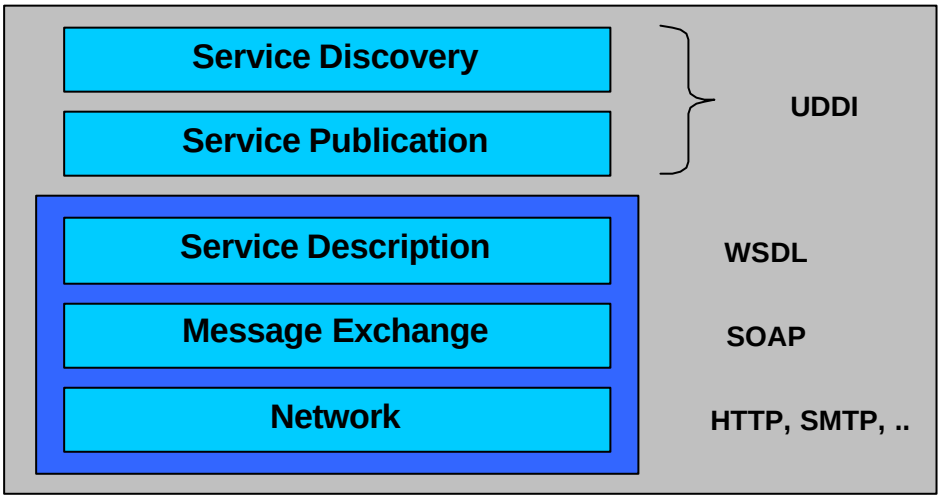
Vor allem im Business-to-Business Geschäftsverkehr konnten Online- und Webservices immer mehr an Bedeutung gewinnen, mit deren Hilfe die Datenverarbeitungssysteme zweier oder mehrerer Firmen direkt miteinander interagieren.
Oft kommen herstellerspezifische Lösungen zum Einsatz, die alle Beteiligten an spezielle Produkte bindet, zumindest muss jedoch den Beteiligten bekannt sein, dass der Service existiert und wie er genutzt werden kann.
Diesem Problem soll UDDI abhelfen. Es handelt sich um ein Middelware-Framework, dass Möglichkeiten zum Veröffentlichen und Auffinden von Web-Diensten bereitstellt, sowie ein Verfahren zur Beschreibung deren Schnittstellen zur Nutzung und Integration.

3 Das UDDI-Framework

3.1 Definitionen

3.1.1 Web Services

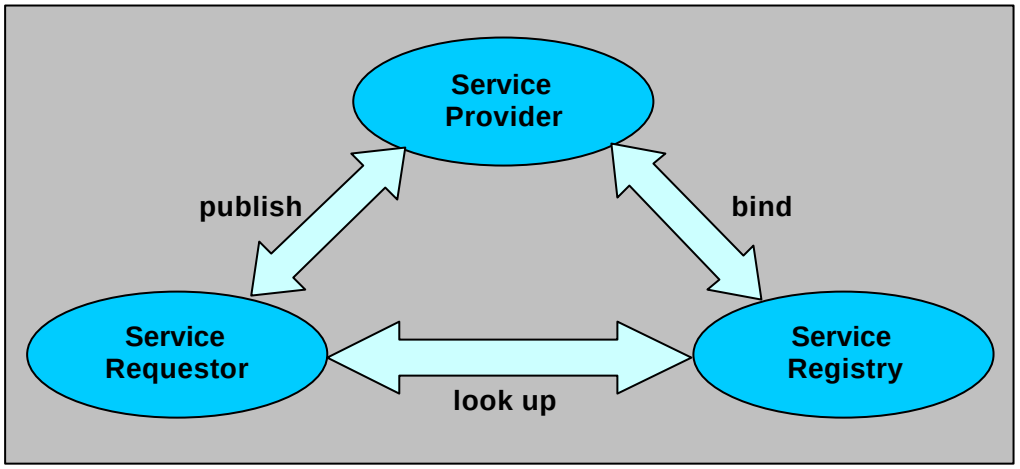
Im allgemeinen sind „Web Services“ Dienste, welche mittels der heutigen Webtechnik über das Internet oder auch Intranet für eine breite Schicht an Nutzern bereitgestellt werden. Ein konkreter Web Service ist eine Anwendung, welche ihre Methoden (Dienstleistungen) über eine Web Schnittstelle zur Verfügung stellt.
Web Services sind heute eine Basis für eine Vielzahl von B2B oder B2C Applikationen.



3.1.2 UDDI: Universal Description Discovery and Integration

Damit ein Web Service genutzt werden kann, muss er zuvor auch bekannt gemacht werden. Ebenso muss es auch möglich sein, ihn im WWW zu finden. Genau hier liegt die Aufgabe von UDDI.
UDDI fungiert als das wesentliche Bindeglied für den Einsatz von Web Services. Das UDDI Framework definiert einen Standard zum Anbieten, Finden und Integrieren von Web Services. Mechanismen zum registrieren von Informationen bzw. zur Suche nach solchen sind in der UDDI API beschreiben.

UDDI greift dabei für die konkrete Beschreibung des Web Service und für den Mitteilungsaustausch über ein Netzwerk auf bekannte Techniken wie z.B. SOAP, WSDL und HTTP zurück. Wichtig ist dabei auch, dass UDDI ein vollkommen Hersteller unabhängiger Standard ist.
Die UDDI Spezifikationen besteht aus einem XML Schema für SOAP Messages und einer Beschreibung der UDDI API Spezifikation.
Hauptbestandteil von UDDI ist die sogenannte UDDI-Registry. In ihr werden alle relevanten Informationen eines Web Service gespeichert und anderen zur Verfügung gestellt. UDDI ist als verteilte Datenbank realisiert, was zur Folge hat, dass ein Dienst nur bei einer Registry bekannt gemacht werden muss, dann aber bei alle Registries zur Verfügung steht.



3.2 Beteiligte Firmen

UDDI wird von der sogenannten UDDI-Community entwickelt, der derzeit 328 Unternehmen angehören.
Sie entwickelte sich aus drei eigenständigen Kooperationen der Firmen IBM, Microsoft und Ariba in den Bereichen XML, SOAP und B2B-Webservices.
UDDI fand schnell auch bei anderen bedeutenden Softwareherstellern Unterstützung, die sich der Community anschlossen. Darunter finden sich bekannte Namen, z.B. SAP, Oracle, Sun, Hewlett-Packard und Compaq.

3.3 Grundlagen

UDDI basiert auf den etablierten und offenen Standards TCP/IP, HTTP, XML und SOAP, die plattformübergreifend und sprachunabhängig zur Verfügung stehen.
Das folgende Schema veranschaulicht, wie UDDI diese Elemente integriert.

Interoperabilitäts-Stack	
	Universal Description, Discovery and Integration (UDDI)
	Simple Object Access Protocol (SOAP)
	Extensible Markup Language (XML)
	Hypertext Transfer Protocol (HTTP)
	Internet-Protokoll (TCP/IP)

Die Themen HTTP und SOAP sind Gegenstand anderer Seminararbeiten, so dass wir im folgenden nur so weit auf die Funktionsweise eingehen, wie es nötig ist, um die Vorteile der jeweiligen Technologie für die Integration in UDDI zu erläutern.

3.3.1 HTTP-Protokoll

UDDI verwendet das Hypertext-Transfer-Protocol als Transportprotokoll zur Übertragung der XML-Nachrichten. Es zeichnet sich für den Einsatz in UDDI durch folgende Eigenschaften besonders aus:

1. Verbreitung im Internet und in den Unternehmensnetzwerken (Verwendung im World-Wide-Web und in Intranets)
2. plattformübergreifende Verwendbarkeit
3. Vorhandene Erfahrung bei Entwicklern und Administratoren
4. Ermöglicht Nutzung von SOAP
5. Optionale Verschlüsselung: HTTPS

Im Gegensatz zu neuen binären Protokollen, minimiert der Einsatz von HTTP daher den Implementierungsaufwand für die Nutzung von UDDI im Unternehmensnetzwerk.
Unter Sicherheitsaspekten betrachtet ist HTTP erprobt und wird von Firewalls gut unterstützt.

3.3.2 SOAP und XML

Das Simple Object Access Protocol[1] (SOAP) ist ein offenes, XML-basiertes Protokoll für den Austausch von Informationen in einer verteilten Umgebung. Es besteht aus drei Teilen:

1. Beschreibung der Daten in der Nachricht und der Art und Weise, wie sie verarbeitet werden sollen
2. Encoding-Regeln für applikationsspezifische Datentypen
3. Konventionen zur Ausführung von entfernten Funktionsaufrufen

SOAP arbeitet Nachrichten-basiert. Die Nachrichten werden codiert im UTF-8-Format über das HTTP-Protokoll übertragen. Dadurch ist SOAP plattformunabhängig, wodurch es sich als Basistechnologie für Webservices allgemein und im speziellen auch für UDDI besonders eignet.

Im folgenden ist die Grundstruktur einer UDDI-SOAP-Nachricht dargestellt.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <UDDI-Funktion UDDI-Namespace-URN Funktions-Attribute>
         ...
         ...
     </UDDI-Funktion>
   </Body>
</Envelope>
```

Blau markiert ist die von SOAP definierte Basisstruktur einer Nachricht, in deren <Body> die UDDI-Funktion eingebettet ist.
„UDDI-Funktion“ bezeichnet stellvertretend eine Funktion aus dem UDDI-API. Zu deren Interpretation wird das UDDI-XML-Schema verwendet, das mittels eines Uniform Resource Names (URN) referenziert wird.

[1] http://www.w3.org/TR/SOAP/

3.3.3 WSDL

Bei der XML-basierten Web Services Description Language (WSDL) handelt es sich um einen W3C-Standard[2] für die Beschreibung von Netzwerk-Diensten.
Sie beschreibt einen Service als eine Menge von Endpunkten, die entweder dokument-orientiert oder prozedural Nachrichten verarbeiten.
Neben einer abstrakten Beschreibung der Verarbeitung und der Nachrichten, definiert WSDL die Endpunkte als eine konkrete Bindung an ein Netzwerkprotokoll mit einem definierten Nachrichtenformat.
WSDL ist also geeignet, die Schnittstelle eines Webservices zu beschreiben. Eine solche Spezifikation wird dazu genutzt, Webservices in Applikationen zu integrieren, was sogar dynamisch zur Laufzeit denkbar ist.

3.4 Komponenten /Architektur

3.4.1 Die UDDI-Registry

Die Registry ist die zentrale Komponente von UDDI.
Die UDDI Business Registry (UBR) wird hauptsächlich auf technischer Ebene genutzt, wobei eine Nutzung auf geschäftlichter Ebene durchaus möglich und sogar sinnvoll ist.
Businessanwender können über einen B2B Marktplatz ein spezielles Such-Portal oder durch Unternehmensanwendungen auf die UDDI Registry zugreifen und so nach Diensten anderer Unternehmen suchen. Entwickler und Techniker können direkt über entsprechende UDDI-Portale auf die UDDI Registry zugreifen, um neue Dienste zu registrieren oder bereits vorhandene Dienste zu bearbeiten.
Standardisierungsgremien, Software-Unternehmen oder Programmierer tragen in der Registry Informationen zu Service-Klassen ein, wobei Unternehmen Informationen über die von ihnen unterstützten und angebotenen Dienste, aber auch öffentliche Informationen über sich selbst registrieren können.

Registration

Informationen können in die UBR entweder mittels eines Web-Interfaces oder durch spezielle Anwendungen, welche das in der UDDI API Spezifikation beschrieben Service Interface nutzen, gespeichert werden. Die UBR entspricht logisch einer zentralen Instanz, ist jedoch physikalisch als ein verteiltes System mit mehreren Knoten realisiert, welche ihre Daten untereinander regelmäßig abgleichen.
Sobald sich ein Unternehmen auf einem Konten mit einem Dienst registriert hat, sind die Daten auf allen anderen Systemen für jeden Anfragenden verfügbar. (register once, published everywhere)
Das zentrale Informations/Datenmodell was von der UDDI Registry benutz wird, ist mittels eines XML Schemas definiert. Die Informationen werden in den sogenannten Pages gespeichert. Es existieren drei verschiedene Arten von Pages, „White Pages“, „Yellow Pages“ und „Green Pages“.

[2] http://www.w3.org/TR/wsdl

White Pages (Namensregister)

Die White Pages stellen ein Namensregister über alle registrierten Anbieter dar. Sie listen die Anbieter mit allen erhältlichen Detail-Informationen, wie Name, Telefon, FAX, URL und genormten Branchen-Beschreibungen alphabetisch sortiert auf. Sie bieten also Informationen über Unternehmen und die von Ihnen angebotenen Dienste, wobei hier die Dienste-Information in textueller Form vorliegt. Hier fließen die Informationen der „businessEntity“ Datenstrukturen ein.

Yellow Pages (Branchenverzeichnis)

Die „Yellow Pages“ sind wie ein Branchenverzeichnis zu sehen.
Hier sind die Unternehmen nach Technologiecodes, Branchentaxonomien oder geographischer Lage geordnet. Die hier gelisteten Daten stammen aus der „businessService Datenstruktur. Die jeweiligen Einträge verweisen auf die zugehörigen White Pages.

Green Pages (Infos zu den Geschäftsmodellen)

Sie enthalten Informationen über die Geschäftsprozesse des Unternehmens. Weiterhin enthalten ist die Definition eines Zugriffspunktes. Ebenso enthalten sind Verweise auf die technische Beschreibung des Dienstes und auf die Beschreibung des Unternehmens.
Hier sind die Daten aus der „bindingTemplate“ Datenstruktur zu finden.

Service Type Registration

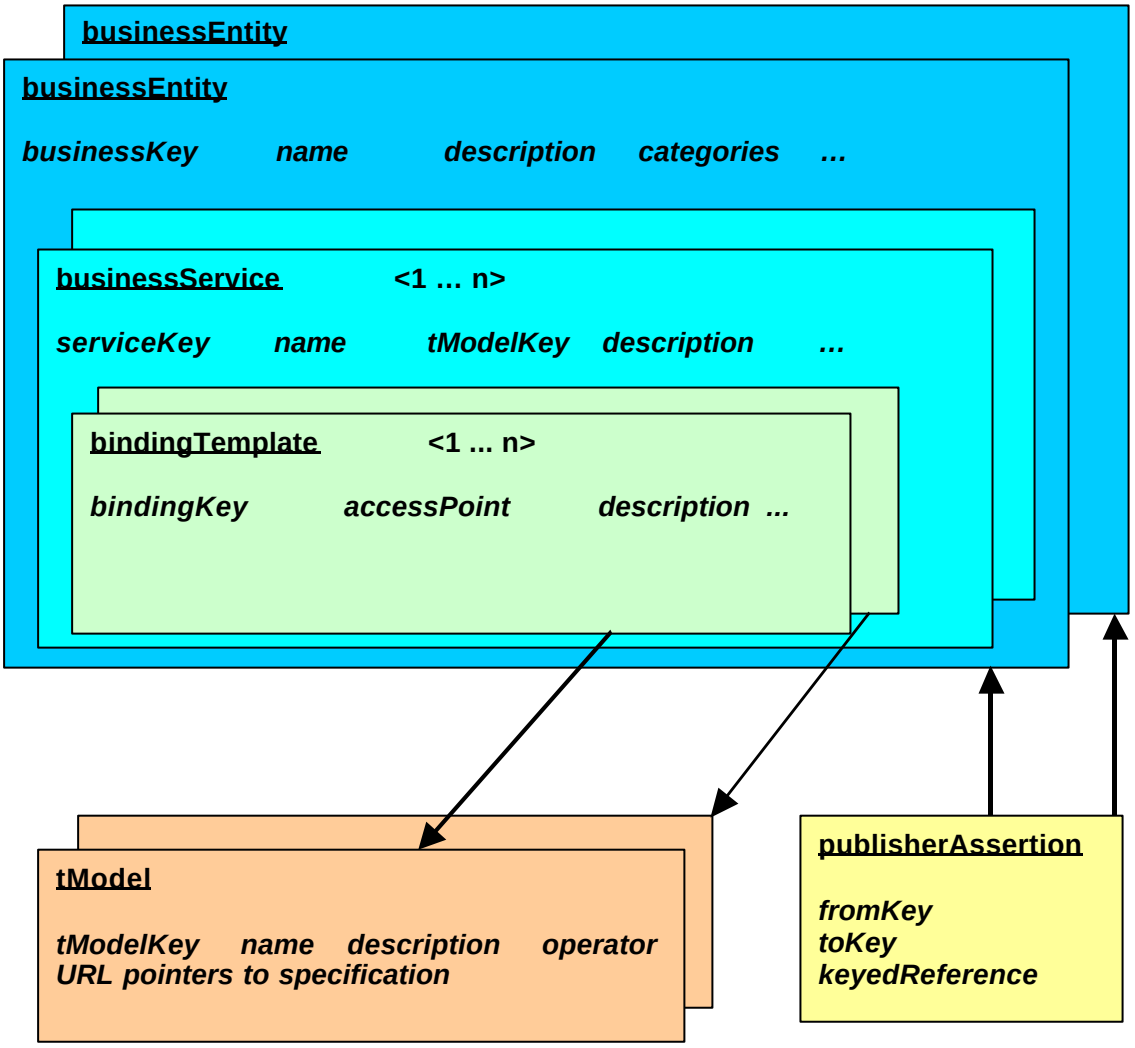
Hier sind die Informationen aus der „tModel“ Datenstruktur zu finden. Dabei handelt es sich um die konkrete technische Beschreibung des angebotenen Dienstes, wie unterstützte Standards, Interchange-Formate, evtl. erforderliche Parameter oder auch Verweise auf weitere evtl. textuelle Beschreibungen der Dienste. Darüber hinaus sind hier auch wieder Verweise auf das anbietende Unternehmen zu finden.

Eindeutige Schlüssel

Die individuellen Daten eines Unternehmens, seine Dienste, technischen Informationen, oder auch nur Informationen über die Spezifikation von Diensten werden separat gehalten und können individuell über einen eindeutigen Schlüssel angesprochen werden. Wenn Informationen erstmalig gespeichert werden, weist die UBR diesen einen eindeutigen Schlüssel zu, der später beim Zugriff auf die Daten benutzt werden kann.
Jeder eindeutige Schlüssel, der von der UBR vergeben wird, hat die Struktur eines „Universal Unique ID“3 (UUID). Es handelt sich hierbei um einen hexadezimalen String, wobei es ausgeschlossen ist, das jemals zwei identische Schlüssel generiert werden.

[3] Die UDDI Struktur und der Algorithmus zur Erzeugung wird in dem ISO/IEC 11578:1996 Standard beschrieben.

3.4.2 Datenstrukturen



Das zentrale Informations-Modell, welches die Registry benutzt, ist in einem XML Schema definiert. Man hat sich entschlossen XML zu benutzen, weil es eine Plattform unabhängige Betrachtung der Daten gewährleistet und eine relativ einfach Beschreibung hierarchischer Beziehungen ermöglicht.
Das UDDI XML Schema definiert 5 zentrale Datenstrukturen, welche alle Informationen bereitstellen, die ein Techniker benötigt, um zu erfahren, wie er die angebotene Dienste eines Unternehmens nutzen kann.
Diese vier Strukturen sind im einzelnen das "businessEntity", der "businessService", das "bindingTemplate", das "tModel" und die „publisherAssertion“.
Jeder dieser XML Strukturen besteht aus einer Reihe von Attributen die entweder eine technische oder wirtschaftliche Beschreibung liefern. Die Strukturen sind in der UDDI API definiert.

Die „businessEntity“ (Business-Informationen)

An der Spitze des gesamten Schemas hängt die Information über das jeweilige Untenehmen. Sie enthält Daten über den jeweiligen Anbieter von Diensten wie z.B. Firmenname, Tel, Fax, URL und Beschreibungsfelder für Adressen, Kategorien, etc. Des weiteren können Sie mehrere Felder für „businessServices“ enthalten.

Spezifikation der XML Struktur

```
<element name = "businessEntity">
    <complexType>
        <sequence>
           <element ref = "discoveryURLs" minOccurs = "0"/>
           <element ref = "name" maxOccurs = "unbounded"/>
           <element ref = "description" minOccurs = "0" maxOccurs = "unbounded"/>
           <element ref = "contacts" minOccurs = "0"/>
           <element ref = "businessServices" minOccurs = "0"/>
           <element ref = "identifierBag" minOccurs = "0"/>
           <element ref = "categoryBag" minOccurs = "0"/>
        </sequence>
        <attribute ref = "businessKey" use = "required"/>
        <attribute ref = "operator"/>
        <attribute ref = "authorized Name">
    </complexType>
</element>
```

Folgende Angaben müssen in der „businessEntity“ Struktur mindestens vorhanden sein:

Feldname	Bedeutung
businessKey	Eindeutiger Identifier einer businessEntity-Instanz
name	Name der registrierten businessEntity-Instanz

Darüber hinaus sind folgende Angaben möglich:

Feldname	Bedeutung
authorizedName	Name desjenigen, welcher den Eintrag erzeugt hat
operator	zertifizierter Name des UDDI-Master-Register(Betreiber der UBR)
description	Textuelle Beschreibung des Unternehmens.
businessServices	Liste einer oder mehrere Dienste Beschreibungen.
identifierBag	Liste von Identifier/Name-Paaren für eine businessEntity-Instanz (z. B. Steuernummer)
categoryBag	Liste von Name/Kategorie-Paaren für eine businessEntity-Instanz (Industrie, Produkt, geographische Codes)

„businessService“ (Service-Informationen)

Diese Datenstruktur ist eine Teil-Struktur der „businessEntity“ und bietet die Möglichkeit, Dienste nach gewissen Merkmalen zu gruppieren. Die zugehörige „businessEntity“ wird über einen businessKey referenziert.

Spezifikation der XML Struktur

```
<element name = "businessService">
    <complexType>
        <sequence>
            <element ref = "name" maxOccurs = "unbounded"/>
            <element ref = "description" minOccurs = "0" maxOccurs = "unbounded"/>
            <element ref = "bindingTemplates"/>
            <element ref = "categoryBag" minOccurs = "0"/>
        </sequence>
        <attribute ref = "serviceKey" use = "required"/>
        <attribute ref = "businessKey"/>
    </complexType>
</element>
```

Folgende Angaben müssen in der „businessService“ Struktur mindestens vorhanden sein:

Feldname	Bedeutung
serviceKey	eindeutiger Identifier eines businessServices, wird von der Registry vergeben
name	Beschreibender Name des Dienstes
bindingTemplate	Liste von bindingTemplates (technische Beschreibung der Dienste-Familie)
businessKey	eindeutiger Identifier einer businessEntity-Instanz

Darüber hinaus sind folgende Angaben möglich:

Feldname	Bedeutung
description	textuelle Beschreibung der Dienste-Familie
categoryBag	Liste von Name/Kategorie-Paaren für eine businessEntity-Instanz (Industrie, Produkt, geographische Codes

„bindingTemplate“

Sie enthalten technische Informationen (Konstruktions-Spezifikationen) zum jeweiligen WebService und den konkreten Zugangspunkt für Serviceanfragen. Im einfachsten Fall kann dies eine einzige URL sein. Auch enthalten sie Referenzen auf „tModels“, welche die Schnittstellen Spezifikation für einen Dienst bestimmen.
Hier wird also genau festgelegt, wie z.B. ein Kaufauftrag auszusehen hat, wie das Protokoll des Auftrages auszusehen hat, welche Sicherheitsmaßnamen erforderlich sind und welche Antwort die Gegenseite auf den Kaufauftrag sendet.
Binding-Templates können beliebig oft in der Service-Information vorhanden sein und haben keine weiteren Teilstrukruren.

Spezifikation der XML Struktur

```
<element name = "bindingTemplate">
    <complexType>
        <sequence>
            <element ref = "description" minOccurs = "0" maxOccurs = "unbounded"/>
            <choice>
                <element ref = "accessPoint" minOccurs = "0"/>
                <element ref = "hostingRedirector" minOccurs = "0"/>
            </choice>
            <element ref = "tModelInstanceDetails"/>
        </sequence>
        <attribute ref = "bindingKey" use = "required"/>
        <attribute ref = "serviceKey"/>
    </complexType>
</element>
```

Folgende Angaben müssen in der „businessService“ Struktur mindestens vorhanden sein:

Feldname	Bedeutung
bindingKey	eindeutiger Identifier eines bindingTemplates (wird von Registry erzeugt), dient zur Identifizierung bei Anfragen und Updates
accessPoint	Startpunkt für die Nutzung des Dienstes, erlaubt ist, was gefällt, URL, Telefonnummern, e-mail-Adressen
hostingRedirector	notwendig, wenn accessPoint-Angabe fehlt, kann z. B. genutzt werden, um 1 accessPoint für mehrere Dienste zentral zu verwalten
tModellInstanceDetails	Daten, die in ihrer Gesamtheit geeignet sind, einen kompatiblen Dienst zu identifizieren
businessKey	eindeutiger Identifier einer businessEntity-Instanz, kann entfallen, wenn die bindingTemplate-Beschreibung Bestandteil einer vollständigen businessEntity-Struktur ist

Darüber hinaus sind folgende Angaben möglich:

Feldname	Bedeutung
Description	textuelle Beschreibung des Dienstes

„tModel“ (Serviceregistrierungen)

Sie dienen als Zeiger auf die eigentlich Spezifikation des Dienstes. Sie enthalten den eindeutigen Schlüssel und Verweisinformationen wie den Spezifikationsnamen, die verantwortliche Stelle oder URL-Verweise auf die Spezifikation selbst.
Das tModel als Spezifikation des Dienstes wird unabhängig von einem bestimmten Anbieter definiert, so dass eine Vielzahl von Anbietern einen gleichartigen Dienst unabhängig voneinander anbieten können.

Spezifikation der XML Struktur

```
<element name = "tModel">
    <complexType>
        <sequence>
            <element ref = "name"/>
            <element ref = "description" minOccurs = "0" maxOccurs = "unbounded"/>
            <element ref = "overviewDoc" minOccurs = "0"/>
            <element ref = "identifierBag" minOccurs = "0"/>
            <element ref = "categoryBag" minOccurs = "0"/>
        </sequence>
        <attribute ref = "tModelKey" use = "required"/>
        <attribute ref = "operator"/>
        <attribute ref = "authorizedName"/>
    </complexType>
</element>
```

Folgende Angaben müssen in der „businessService“ Struktur mindestens vorhanden sein:

Feldname	Bedeutung
tModelKey	eindeutiger Identifier einer tModel-Struktur (wird von der Registry erzeugt), muss bei Updates angegeben werden
name	Name des Dienstes, sollte ihn eindeutig kennzeichnen

Darüber hinaus sind folgende Angaben möglich:

Feldname	Bedeutung
description	textuelle Beschreibung des Dienstes
identifierBag	Liste von Identifier/Name-Paaren für eine tModel-Instanz
categoryBag	Liste von Name/Kategorie-Paaren für eine tModel-Instanz (Industrie, Produkt, geographische Codes)
overviewDoc	Referenz auf eine Remote-Beschreibung des Dienstes

„publisherAssertion“

Die „publisherAssertion“ Struktur wird benutzt, um Beziehungen zwischen mehreren Unternehmen darzustellen. Es ist z.B. sehr schwierig, große Konzerne in einer einzigen „businessEntity“ abzubilden. Darum werden mehrere „businessEntities“ veröffentlicht, um individuelle Teile eines großen Konzerns darzustellen. Um nun die Beziehung für andere sichtbar zu machen, wird die „publisherAssertion“ benutzt.

Spezifikation der XML Struktur

```
<element name = "publisherAssertion">
    <complexType>
        <sequence>
            <element ref = "fromKey"/>
            <element ref = "toKey"/>
            <element ref = "keyedReference"/>
        </sequence>
    </complexType>
</element>
```

Folgende Angaben müssen in der „publisherAssertion“ Struktur mindestens vorhanden sein:

Feldname	Bedeutung
fromKey	Eindeutige Referenz auf die erste businessEntity
toKey	Eindeutige Referenz auf die zweite businessEntity
keyedReference	Bestimmt den Typ einer Beziehung, für welche die Assertion veröffentlicht wurde, durch einen tModelKey und durch die Beschreibung mittels eines keyName / keyValue Paares

Security

Nur autorisierten Nutzern ist es erlaubt, mittels der Publisher API Informationen in die UBR einzutragen oder zu ändern.
Jede Implementation der verteilten UBR verwaltet selbstständig eine einzigartige Liste von autorisierten Parteien und protokolliert, welche "businessEntity" oder welches "tModell" von wem erzeugt wurde.
Änderungen sind nur erlaubt, wenn die Anfrage zur Änderung mittels API Call von demjenigen kommt, welcher die betroffene Informationen eingestellt hat.
Jeder Instanz einer UBR, auch "Operator Site" genannt, ist es erlaubt, eigene Authentifizierungs- Mechanismen zu implementieren. Dies müssen aber mindestens einem gemeinsamen Sicherheits-Standard entsprechen und einen gewissen Schutz bieten. Meistens werden sogenannte „Authentication Tokens“ von der Operator-Site vergeben, welche nur von der ausgebenden Stelle verstanden und akzeptiert werden. Vorher kann jedoch noch ein Login verlangt werden.
Die Sicherheit auf der Übertragungsstrecke wird mittels SSL oder https gewährleistet.

3.4.3 UDDI-API

UDDI definiert ein API[4] zum Zugriff auf die in der UDDI-Registry enthaltenen Informationen. Es lässt sich in zwei Bereiche untergliedern:

1. Inquery-API: Funktionen zum Auffinden von Informationen
2. Publisher-API: Funktionen zum Veröffentlichen und Verwalten veröffentlicher Informationen

Der UDDI-Funktionsaufruf wird im <Body> der SOAP-Nachricht an die Registry übertragen (s. 2.3.2).
Er ist wie folgt aufgebaut:

```
<Funktionsname Version-ID [ Attribut ] Namespace-URN >
    <Argumentname_1>Wert_von_Argument_1</Argumentname_1>
    <Argumentname_2>Wert_von_Argument_2</Argumentname_2>
        .
        .
        .
    <Argumentname_n>Wert_von_Argument_n</Argumentname_n>
</Funktionsname >
```

Feldname	Bedeutung
`Funktionsname`	Name der Funktion, erläutert in 2.4.2.1 und 2.4.2.2
`Version-ID`	Versions-Attribut, dass die verwendete UDDI-Version spezifiziert. *generic="1.0":* UDDI Version 1.0 *generic="2.0":* UDDI Version 2.0
`Attribut`	Attribute beeinflussen die Funktionsausführung bzw. qualifizieren den Rückgabewert Beispiel: *maxRows*: limitiert die Anzahl der Rückgabewerte *truncated*: zeigt an, dass es sich bei den Rückgabewerten nicht um alle Ergebnisse der Anfrage handelt
`Namespace-URN`	bezeichnet den UDDI-XML-Namespace. Die Namespace-Auswahl erfolgt in Abhängigkeit der Versions-ID: *xmlns="urn:uddi-org:api"*: UDDI-Version 1.0 *xmlns="urn:uddi-org:api_v2"*: UDDI Version 2.0
`Argumentname_n`	Name eines Funktionsarguments. Welche Argumente für eine Funktion zulässig sind, spezifiziert das UDDI-API. Beispiel: *name*: Übergibt der Funktion einen Parameter namens „name“, der als Wert einen String enthält, nach dem Namensinformationen der jeweiligen „Page“ der Registry durchsucht werden sollen.

[4] http://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf

Die folgende Auflistung der API-Funktionen soll zum besseren Verständnis der Abläufe in Kapitel 3 beitragen und einen kurzen Einblick in den Umfang der UDDI-Spezifikation geben.

Inquiry API

V.	Funktion Argumente[5]	Rückgabetyp	Beschreibung
1, 2	`find_binding`		Sucht Binding-Informationen innerhalb eines businessService, der durch dem serviceKey referenziert wird
	`serviceKey` `tModelBag`	`bindingDetail`	
1, 2	`find_business`		Sucht Webservices-Anbieter nach angegebenen Auswahlkriterien

	`[name]`
	`[discoveryURLs]`
	`[tModelBag]`
	`...`
		`businessList`	

2	`find_relatedBusiness`		Sucht Webservice-Anbieter, die mit einem gegebenen Anbieter in Beziehung stehen, der mit dem businessKey referenziert wird
	`businessKey`	`relatedBusinessesList`	
1, 2	`find_service`		Sucht einen Service nach gegebenen Kriterien innerhalb der Beschreibung eines Webservice-Anbieters, der durch den businessKey referenziert wird

	`businessKey`
	`[name]`
	`[categoryBag]`
	`[tModelBag]`
		`serviceList`	

1, 2	`find_tModel`		Sucht eine tModel-Struktur nach gegebenen Kriterien

	`[name]`
	`[categoryBag]`
	`[tModelBag]`
		`tModelList`	

1, 2	`get_bindingDetail`		Liefert Binding-Detail-Informationen zu einem Service, der über durch den bindingKey indirekt referenziert wird. Diese Informationen werden zur Nutzung des Webservice benötigt
	`bindingKey`	`bindingDetail`	
1, 2	`get_businessDetail`		Liefert zu Webservice-Anbietern Detailinformationen (businessEntity)
	`businessKey`	`businessDetail`	
1, 2	`get_businessDeatilExt`		Liefert zu Webservice-Anbietern erweiterte Informationen
	`businessKey`	`businessDetailExt`	
1, 2	`get_serviceDetail`		Liefert zu Services (eines

[5] Hier sind nur die wichtigsten Argumente aufgeführt.

V.	Funktion Argumente[5]	Rückgabetyp	Beschreibung
	serviceKey	serviceDetail	Anbieters oder mehrerer Anbieter) Detailinformationen (u.a. bindingKey)
1, 2	get_tModelDetail		Liefert zu einer oder mehreren tModel-Struktur(en) Detailinformationen
	tModelKey	tModelDetail	

Publisher API

V.	Funktion Argumente[6]	Rückgabetyp	Beschreibung
2	add_publisherAssertions		Fügt Beziehungen zwischen zwei Webservice-Anbietern hinzu.
	authInfo publisherAssertion	dispositionReport	
1, 2	delete_binding		Entfernt Binding-Informationen (bindingTemplate), referenziert durch den gegebenen bindingKey aus der Beschreibung eines Service (businessService)
	authInfo bindingKey	DispositionReport	
1, 2	delete_business		Entfernt Informationen über einen Webservice-Anbieter (businessEntity) aus der Registry
	authInfo businessKey	dispositionReport	
2	delete_publisherAssertions		Löscht eine existierende Beziehung zwischen zwei Webservice-Anbietern
	authInfo publisherAssertion	dispositionReport	
1, 2	delete_service		Entfernt einen Service (serviceTemplate), referenziert durch einen serviceKey aus den Beschreibungen eines Webservice-Anbieters (businessEntity)
	authInfo serviceKey	dispositionReport	
1, 2	delete_tModel		Versteckt über ein tModel registrierte Informationen.
	authInfo tModelKey	dispositionReport	
1, 2	discard_authToken		„Abmelden“ bei der Registry; das verwendete Token wird ungültig
	authInfo	dispositionReport	
2	get_assertionStatusReport		Liefert Beziehungsdefinitionen,

[6] Hier sind nur die wichtigsten Argumente aufgeführt

V.	Funktion Argumente[b]	Rückgabetyp	Beschreibung
	authInfo completionStatus	assertionStatusReport	die noch bestätigt werden müssen
1, 2	get_authToken		„Anmelden“ bei der Registry Als Parameter werden Benutzername (userID) und Passwort (cred) übergeben. Man erhält eine Struktur, die bei allen anderen Publisher-API-Funktionen zur Authentisierung übergeben werden muss
	userID cred	authToken (enthält authInfo)	
2	get_publisherAssertions		Liefert existierende Beziehungsdefinitionen
	authInfo	publisherAssertions	
1, 2	get_registeredInfo		Liefert eine Liste mit Business- und tModel-Daten, die der angemeldete Benutzer bei der Registry veröffentlicht hat
	authInfo	registeredInfo	
1, 2	save_binding		Legt Binding-Informationen zu angebotenen Webservices an oder ändert diese. Die übergebenen bindingTemplate-Strukturen enthalten den Key des zugehörigen Service
	authInfo bindingTemplate	bindingDetail	
1, 2	save_business		Legt Informationen über Webservice-Anbieter (businessEntity) an bzw. ändert diese. „Broadcast“ aller save_xxx-Funktionen Version 2: Kann zusätzlich Referenz auf übergeordnete businessEntity anlegen
	authInfo businessEntity	businessDetail	
1, 2	save_service		Legt Informationen über Services an bzw. ändert diese. Die übergebenen businessService-Strukturen enthalten den businessKey des zugehörigen Anbieters
	authInfo businessService	serviceDetail	
1, 2	save_tModel		Legt tModel-Strukturen an bzw. ändert diese.
	authInfo tModel	tModelDetail	
2	set_publisherAssertions		Legt eine vollständige

V.	Funktion		Beschreibung
	Argumente[b]	Rückgabetyp	
	`authInfo` `publisherAssertion`	`publisherAssertions`	Beschreibung aller sichtbaren Beziehungen zwischen Webservice-Anbietern an bzw. ersetzt vorhandene. Solche Beziehungen sollen über das Inquiry-API sichtbar sein (find_relatedBusiness). Solche Beziehungen müssen vom Partner bestätigt werden, in dem er die gleiche Beziehung anlegt.

Code-Beispiel

Folgendes in Java geschriebene Beispiel veranschaulicht die Verwendung des UDDI-API. Ein Beispielhafter Funktionsaufruf, *find_business*, wird an eine UDDI-Registry gerichtet. Es sollen max. 50 Webservice-Anbieter zurückgeliefert werden, deren Name den String „Weather“ enthält. Als zusätzliche Parameter werden Sortieranweisungen übergeben.
Der String *message* enthält den Funktionsaufruf, eingebettet in die umschließende SOAP-Nachricht. Er wird UTF-8-codiert über eine HTTP-Verbindung zur Registry gesendet. Die Registry führt die UDDI-Funktion aus und sendet eine SOAP-Nachricht mit dem Ergebnis, eine *businessInfos*-Struktur, zurück.

```
/*
 * uddi.java
 * Author: Chr. Ersfeld, R. Kortas
 */

import java.net.*;
import java.io.*;

public class uddi {

    public static void main(String args[]) throws Exception {
        System.out.println("UDDI-Message Demoprogramm");

        String message = new String(
            "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>"
           +"<Envelope xmlns=\"http://schemas.xmlsoap.org/soap/envelope/\">"
           +"<Body>"
           +"<find_business xmlns=\"urn:uddi-org:api\" generic=\"1.0\" maxRows=\"50\">"
           +"<findQualifiers><findQualifier>caseSensitiveMatch</findQualifier>"
           +"<findQualifier>sortByNameDesc</findQualifier>"
           +"</findQualifiers>"
           +"<name>Weather</name>"
           +"</find_business>"
           +"</Body>"
           +"</Envelope>"
        );

        System.out.println("Funktionsaufruf an die Registry:\n"+message);

        // konfiguriere HTTP-Verbindung
        URL url = new URL("http://www-3.ibm.com/services/uddi/inquiryapi");
        URLConnection con = url.openConnection ();

        con.setDoInput(true);
        con.setDoOutput(true);
        con.setUseCaches(false);
        con.setAllowUserInteraction(false);

        con.setRequestProperty("Content-Type", "text/xml; charset=\"utf-8\"");
        con.setRequestProperty("Accept", "text/xml");
        con.setRequestProperty("SOAPAction", "\"\"");

        // sende Anfrage
        OutputStreamWriter out = new OutputStreamWriter(con.getOutputStream(), "UTF-8");
        out.write(message, 0, message.length());
        out.flush();

        // empfange Antwort
        InputStreamReader in = new InputStreamReader(con.getInputStream(), "UTF-8");
        StringBuffer answer = new StringBuffer();
        int raw;
        while((raw = in.read()) > 0)
            answer.append((char)raw);

        in.close();
        out.close();
        System.out.println("Antwort der Registry:\n"+answer);

    }

}
```

Ausgabe:

```
UDDI-Message Demoprogramm
Funktionsaufruf an die Registry:
<?xml version="1.0" encoding="UTF-8" ?><Envelope
xmlns="http://schemas.xmlsoap.org/soap/envelope/"><Body><find_business
xmlns="urn:uddi-org:api" generic="1.0"
maxRows="50"><findQualifiers><findQualifier>caseSensitiveMatch</findQualifi
er><findQualifier>sortByNameDesc</findQualifier></findQualifiers><name>Weat
her</name></find_business></Body></Envelope>
Antwort der Registry:
<?xml version="1.0" encoding="UTF-8" ?><Envelope
xmlns="http://schemas.xmlsoap.org/soap/envelope/"><Body><businessList
generic="1.0" xmlns="urn:uddi-org:api" operator="www.ibm.com/services/uddi"
truncated="false"><businessInfos><businessInfo businessKey="90C0E220-32CE-
11D6-87CE-000C0E00ACDD"><name>Weatherbee Consulting</name><description
xml:lang="en">Training, Mentoring and Consulting: WebSphere and
J2EE</description><serviceInfos></serviceInfos></businessInfo><businessInfo
businessKey="92BF57B3-4FFB-4B8E-8102-
E84ED794DCAB"><name>WeatherBusinessTest</name><description
xml:lang="en">Temporary Business to test weather
report</description><serviceInfos><serviceInfo serviceKey="878539C9-45B8-
48B9-B8D8-94A98101B387" businessKey="92BF57B3-4FFB-4B8E-8102-
E84ED794DCAB"><name>Weather</name></serviceInfo></serviceInfos></businessIn
fo><businessInfo businessKey="85FC4F5A-AB34-4AB6-97ED-
3607E1847840"><name>Weather Provider</name><description
xml:lang="en">Provides local
temprature</description><serviceInfos><serviceInfo serviceKey="D6753BCD-
96BB-47E5-8C9F-E28AE7319ABB" businessKey="85FC4F5A-AB34-4AB6-97ED-
3607E1847840"><name>Weather_Service</name></serviceInfo></serviceInfos></bu
sinessInfo></businessInfos></businessList></Body></Envelope>
```

4 UDDI in der Anwendung

4.1 Überblick

Der Anbieter eines Webservices verwendet die Funktionen des Publisher-API, um seinen Webservice bekannt zu machen. Er beschreibt, wie der Service technisch genutzt werden kann in den dafür vorgesehenen Datenstrukturen.
Der Nutzer kann mit den Funktionen des Inquiry-API die UDDI-Registry nach Anbietern, angebotenen Diensten und technischen Informationen zur Nutzung und Integration durchsuchen.

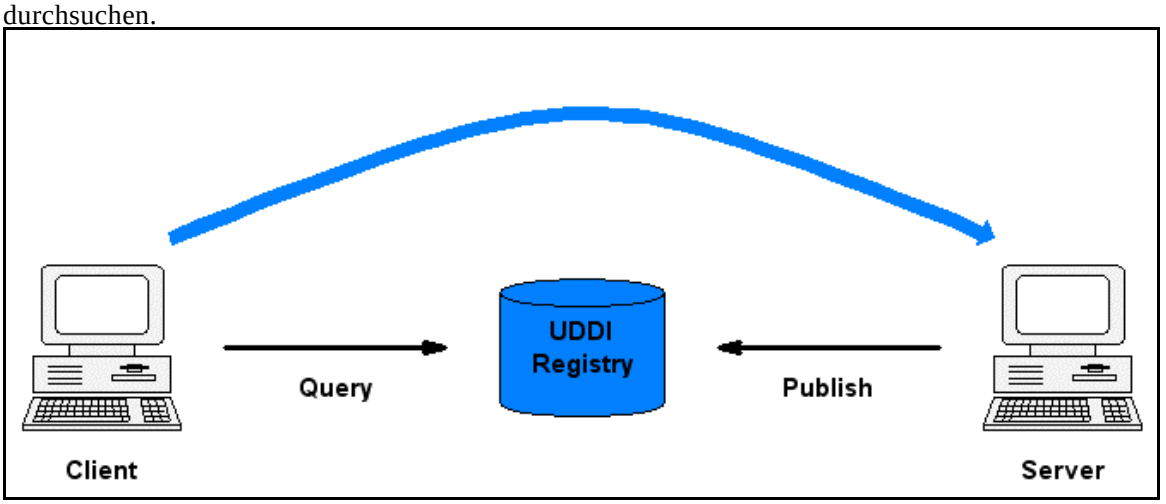


Abbildung 1: Kommunikation über UDDI

4.2 Webservices anbieten

Um ein „Business“ veröffentlichen zu können, benötigt man eine Benutzerkennung und zugehöriges Passwort von einem Registry-Betreiber. Diese Daten erhält man, nachdem man als Benutzer von dem jeweiligen Betreiber registriert worden ist.
Die Zugangsdaten ermöglichen nur den verändernden Zugriff auf der Registry, von der man die Zugangsdaten erhalten hat. Die vorgenommenen Veränderungen gleichen die Registries verschiedener Betreiber wie bereits erläutert selbständig untereinander ab.

Für den verändernden Zugriff kann entweder ein Programm verwendet werden, dass das Publisher-API implementiert oder ein Web-Interface, das fast alle Regitry-Betreiber anbieten.

Aufgrund der Datenhierarchie gibt es verschiedene Wege, Informationen über einen Webservice zu veröffentlichen. So subsumiert die Funktion `save_service` die Funktion `save_binding`, `save_business` wiederum subsumiert `save_service` und ist geeignet, mit einem Funktionsaufruf alle Daten dieses Anbieters mit nur diesem Funktionsaufruf zu veröffentlichen.

Im folgenden Beispiel soll den veröffentlichten Daten eines Webservice-Anbieters die Beschreibung eines weiteren Service hinzugefügt werden.
Für diesen Service gibt es bereits eine WSDL-Spezifikation

(Beispiel)

beteiligte Funktionen	erhaltene Daten	Ausführungsschritt
ist bekannt	`businessKey`	
`get_authToken`	`authToken`	Der Administrator der veröffentlichten Daten meldet sich bei der Registry an und erhält eine `authInfo`-Struktur innerhalb von `authToken`, die er bei jeder daten-verändernden Funktion als Argument angeben muss
	WSDL-Spezifikation	Der Programmierer des Webservices hat eine WSDL-Spezifikation angefertigt.
über Webserver bereitstellen	*URL der WSDL Spezifikation*	Diese wird für Nutzer des Webservices über das Internet/Intranet verfügbar gemacht.
integriert URL d. WSDL-Spezifikation	`tModel`	Sofern es noch kein tModel für den Service gibt, wird eine entsprechende Struktur angelegt.
`save_tModel`	`tModelDetail`	Sie wird in der Registry gespeichert und somit veröffentlicht.
enthalten in tModelDetail	`tModelKey`	Im Rückgabewert ist bei der neue Schlüssel enthalten, der dieses tModel referenziert.
	`z.B. accessPoint`	Notwendige Informationen zur Integration bzw. Nutzung des Webservice werden zusammengestellt
referenziert tModelKey	`bindingTemplate`	Diese Informationen werden im bindingTemplate zusammengefasst
integriert bindingTemplate referenziert businessKey	`businessService`	Die businessService Struktur wird angelegt. Sie enthält das bindingTemplate, sowie eine Bezeichnung und Kurzbeschreibung des Service
`save_service`	`serviceDetail`	Die businessService Struktur wird in der Registry gespeichert und damit veröffentlicht

4.2.1 Codebeispiel

s. Anhang

4.3 Webservices suchen und nutzen

Jeder einzelne angebotene Webservice wird in technischer Hinsicht durch eine bindingTemplate Struktur beschrieben. Darüber hinaus kann UDDI auch auf weitergehende Spezifikationen, z.B. WSDL, verweisen. Die darin enthaltenen Informationen benötigt ein Programmierer, um einen Webservice in seine Applikation zu integrieren. Er kann sich diese Informationen wie folgt beschaffen:

(Beispiel)

beteiligte Funktionen	erhaltene Daten	Ausführungsschritt
`find_business` ↘ `get_businessDetail`	`businessEntity`	Der Programmierer eines Programms, dass den Webservice nutzen soll, sucht mit Hilfe eines Webinterface oder eines Programms, das das Inquiry-API implementiert, nach dem Anbieter des Service.
enthalten in businessEntity	`serviceKey`	Er sucht den gewünschten Service aus der Liste der von diesem Anbieter bereitgestellten Services aus.
`get_ServiceDetail`	`serviceDetails`	Der Programmierer ruft Detailinformationen zum Service ab.
enthalten in serviceDetails	`bindingTemplate`	Im bindingTemplate sind bereits wichtige Informationen zur technischen Integration erhalten
enthalten in bindingTemplate	`z.B. accessPoint`	Sie werden so weit für die Applikation erforderlich verwendet
enthalten in bindingTemplate	`tModelKey`	Unter der zugehörigen tModel-Beschreibung finden sich weitere Informationen. Daher extrahiert der Programmierer den tModelKey zur weiteren Verwendung
`get_tModelDetails`	`tModelDetail`	Der Programmierer ruft die zum tModel gehörigen Informationen ab.
enthalten in tModelDetail	*URL einer WSDL-Spezifikation*	Die tModelDetail Struktur kann einen Link auf eine WSDL-Spezifikation enthalten, die den Service im Detail beschreibt.
http get	*WSDL Spezifikation*	Der Programmierer ruft die WSDL-Spezifikation ab
enthalten in WSDL-Spezifikation	*z.B. Argument-Typen Rückgabetypen*	Er extrahiert die benötigten Informationen, z.B. Parameterformate und Rückgabetypen, die er benötigt, um den Service in seine Applikation zu integrieren.

4.3.1 Code-Beispiel

s. Anhang

4.4 UDDI-Tools

Es existieren mittlerweile ein Vielzahl von kommerziellen und freien Toolkits und Entwicklungsumgebungen. Sie ermöglichen es mittels spezieller APIs auf UDDI Business Registrys zuzugreifen und sogar seine eigene Registry aufzubauen.
Will man sich nicht mit den UDDI APIs auseinandersetzen und verfügt man über keine XML Kenntnisse, so ist das Arbeiten mit UDDI trotzdem möglich. Es existieren Web-Frontends, z.B. von IBM, mit denen es möglich ist, Business Daten zu registrieren und nach Diensten zu suchen.
Hier nun eine kleine Auswahl von Toolkits, um auf UDDI Registries zugreifen zu können:

4.4.1 JUDDI

Das Open Source Projekt JUDDI (www.juddi.org) wurde auf Basis von Java 1.2.2 implementiert und ermöglicht es, geeignete Frontends zu erstellen, um in den UDDI Registries zu suchen und Web-Services zu registrieren.

4.4.2 UDDI4J

Das Projekt UDDI für Java (UDDI4J) wurde im Februar 2001 von IBM für den Open Source Bereich freigegeben.
Bei dem UDDI4J-Softwaremodul handelt es sich um ein Java API, das ein Programmier-Interface auf Client-Seite zur Verfügung stellt, mit dem man auf einfache Art und Weise auf jede UDDI Registry zugreifen kann. Das UDDI4J-Paket kann beispielsweise eingesetzt werden, um Java-Applikationen zu schreiben, die Publish- und Find-Funktionen in einer UDDI Registry ermöglichen. Dabei wird der Client-Code aus dem Apache SOAP Projekt eingesetzt, um Messages zwischen Client und dem UDDI Registry zu verarbeiten.
Der Code kann kostenlos und online über die IBM developerWorks-Site bezogen werden. (http://www-124.ibm.com/developerworks/oss/uddi4j/)

4.4.3 IBM Web-Sevices Toolkit

UDDI4J ergänzt auch das Web Services Toolkit von IBM (WSTK), das Programmierern eine Runtime-Umgebung und Beispiele für das Design sowie für die Ausführung von Web Services zur Verfügung stellt. Das WSTK ist seit Juli 2000 lieferbar und kann über die alphaWorks Website von IBM bezogen werden
www.alphaworks.ibm.com/tech/webservicestoolkit
Mit dem WSTK erhält man alle Tools, die man zum Aufbau einer Service-orientierten Architektur braucht: Neben Axis(SOAP, wsdl2java, java2wsdl) bekommt man auch einen Registry-Server, mit dem man die UDDI-Api's im Intranet zu Hause ausprobieren kann.

4.4.4 pUDDIng

Bei pUDDIng handelt es sich um eine weitere Open Source Implementierung für die UDDI Spezifikation V2.0. Sie beinhaltet sowohl einen Client als auch einen Oracle basierten Server

5 Ausblick

Trotz der breiten Unterstützung bei den Softwareherstellern hat sich UDDI bislang noch nicht in dem Maße durchsetzen können, wie man es erwartet hatte.
Das liegt nicht zuletzt daran, dass UDDI für die derzeit verbreitetsten Applikationen im Business-to-Business Umfeld kaum Vorteile bringt.
Betrachtet man den Einsatz von Webservices als gerade begonnenen weiteren Schritt in der Evolution des Internets, so bietet UDDI die Grundlage für die weitere Entwicklung.
Mit der Implementierung der Business-Registries haben die Softwarehersteller die nötigen Voraussetzungen geschaffen. Den Entwicklern steht bereits eine grosse Zahl an Werkzeugen zur Verfügung. Es ist also davon auszugehen, dass UDDI in Zukunft an Bedeutung gewinnen wird.

6 Literatur

UDDI Executive White Paper	Ariba, IBM, Microsoft http://www.uddi.org/pubs/UDDI_Executive_White_Paper.PDF
UDDI Technical White Paper	Ariba, IBM, Microsoft http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.PDF
UDDI Version 2.0 API Specification	Ariba, IBM, Microsoft et al. http://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf
UDDI Version 2.0 Data Structure Reference	IBM, Microsoft, SAP et. al. http://www.uddi.org/pubs/DataStructure-V2.00-Open-20010608.pdf
UDDI Programmer's API 1.0	Ariba, IBM, Microsoft http://www.uddi.org/pubs/ProgrammersAPI-V1.01-Open-20010327_2.pdf
UDDI Data Structure Reference V1.0	Ariba, IBM, Microsoft http://www.uddi.org/pubs/DataStructure-V1.00-Open-20000930_2.pdf
HTTP - Hypertext Transfer Protocol	W3C, HTTP Working Group http://www.w3.org/Protocols/
SOAP Version 1.2	W3C, XML Protocol Working Group http://www.w3.org/2000/xp/Group/
Web Services Description Language (WSDL) 1.1	W3C, Ariba, IBM, Microsoft http://www.w3.org/TR/wsdl
„Weiß, Gelb, Grün UDDI: Geschäftsdaten bereitstellen und nutzen“	Endlich, Stefan iX 2/2001, Heise Verlag

7 Anhang

Die folgenden Code-Beispiele verwenden das UDDI4J API.

7.1 RegisterBusiness

Das Code-Beispiel „RegisterBusiness.java“ registriert in der IBM Test Registry dir Firma „KONECO Systems“.
Um auf der Test Registry arbeiten zu können, muss man sich zuvor registrieren lassen, wodurch man eine UserID und ein Paßwort erhält. Diese muss man dann bei der Speicherung eines Unternehmens mit angeben. Bei der Registrierung wird eine verantwortliche Person „Reto Kortas“ und eine Beschreibung „IT-Dienstleistungen aller Art“ angegeben. Als Rüchgabewert nach der Speicherung erhält man ein „BusinessDetail“ Objekt, welches unter anderem den durch die UBR vergebenen „BusinessKey“ enthält.

```
import org.uddi4j.*;
import org.uddi4j.client.*;
import org.uddi4j.datatype.*;
import org.uddi4j.datatype.assertion.*;
import org.uddi4j.datatype.binding.*;
import org.uddi4j.datatype.business.*;
import org.uddi4j.datatype.service.*;
import org.uddi4j.datatype.tmodel.*;
import org.uddi4j.request.*;
import org.uddi4j.response.*;
import org.uddi4j.util.*;

import org.w3c.dom.Element;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.util.Vector;
import java.util.Properties;
import java.io.*;

public class RegisterBusiness {
    //addresses of the IBM TestRegistry
    String inquiryURL =
       "http://www-3.ibm.com/services/uddi/testregistry/inquiryapi";
    String publishURL =
       "https://www-3.ibm.com/services/uddi/testregistry/protect/publishapi";
    //my personal Data for using the TestRegistry
    final String USERID = "koneco";
    final String PASSWD = "test1234";
    final String BKEY="3806B9F0-7BAA-11D6-AC1C-000C0E00ACDD";
    //the name of my business which I want to register
    final String businessName = "KONECO Systems";

    public static void print(String messageToPrint){
        System.out.println("\n"+messageToPrint+"\n");
    }

    public static void main (String args[]) {
        RegisterBusiness myBusiness= new RegisterBusiness();
        myBusiness.run();
        System.exit(0);
    }

    public void run() {
        // set-up a new UDDIProxy object
        UDDIProxy testProxy = new UDDIProxy();
```

```
        try {
            testProxy.setInquiryURL(inquiryURL);
            testProxy.setPublishURL(publishURL);

            // Get an authorization token from the UBR
            print("** Get Authentication Token from the Registry **");
            AuthToken token = testProxy.get_authToken(USERID,PASSWD);
            print("Returned Token: "+ token.getAuthInfoString());
            print("Operator of the UBR:"+token.getOperator());

            Vector entities = new Vector();
             /*Create a new business entity. BusinessKey must be "" to save a new
             business */
            print("Creating new BusinessEntity !");
            BusinessEntity be = new BusinessEntity("",businessName);
            be.setAuthorizedName("Reto Kortas");
            be.setDefaultDescriptionString("IT-Dienstleistungen aller Art");
            entities.addElement(be);

             /* Save business: The BusinessDetail object contains the final results of
             the call that reflects the new registered information for the
             businessEntity.*/

            print("Save new Business in the UBR !");
            BusinessDetail bd =
                   testProxy.save_business(token.getAuthInfoString(),entities);

            // Process returned BusinessDetail object
            Vector businessEntities = bd.getBusinessEntityVector();
            BusinessEntity returnedBusinessEntity =
                   (BusinessEntity)(businessEntities.elementAt(0));
             print("Returned businessKey: "+
                   returnedBusinessEntity.getBusinessKey());
            print("Returned Operator: "+returnedBusinessEntity.getOperator());
      }
             // catch all UDDI Exceptions
             catch (UDDIException e) {
               DispositionReport dr = e.getDispositionReport();
                if (dr!=null) {
               System.out.println("UDDIException faultCode:" + e.getFaultCode() +
                                  "\n operator:" + dr.getOperator() +
                                  "\n generic:"  + dr.getGeneric() +
                                  "\n errno:"    + dr.getErrno() +
                                  "\n errCode:"  + dr.getErrCode() +
                                  "\n errInfoText:" + dr.getErrInfoText());
               }
            e.printStackTrace();
             }
             // catch regular Exceptions
             catch (Exception e) {
             e.printStackTrace();
             }
      }
}
```

7.2 AddContacts

Hier werden einer bereits zuvor registrierten Firma (BusinessEntity) Kontakt-Informationen hinzugefügt. Alle Informationen sind in einem „Contacts“ Objekt gekapselt. Dieses kapselt ein „Contact“ Objekt, welches dann die eigentlichen Informationen (Phone, Name, ...)enthält. Die BusinessEntity, zu welcher die Informationen hinzugefügt werden sollen, wir über den bei der Erzeugung erhaltenen BusinessKey referenziert.

```
import org.uddi4j.*;
import org.uddi4j.client.*;
import org.uddi4j.datatype.*;
import org.uddi4j.datatype.assertion.*;
import org.uddi4j.datatype.binding.*;
import org.uddi4j.datatype.business.*;
import org.uddi4j.datatype.service.*;
import org.uddi4j.datatype.tmodel.*;
import org.uddi4j.request.*;
import org.uddi4j.response.*;
import org.uddi4j.util.*;

import org.w3c.dom.Element;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.util.Vector;
import java.util.Properties;
import java.io.*;


public class AddContacts {
    //addresses of the IBM TestRegistry
    String inquiryURL =
       "http://www-3.ibm.com/services/uddi/testregistry/inquiryapi";
    String publishURL =
       "https://www-3.ibm.com/services/uddi/testregistry/protect/publishapi";
    //my Data for using the TestRegistry
    final String USERID = "koneco";
    final String PASSWD = "test1234";
    final String BKEY="3806B9F0-7BAA-11D6-AC1C-000C0E00ACDD";
    //the name of the business to register
    final String businessName = "KONECO Systems";

    public static void print(String messageToPrint){
        System.out.println(messageToPrint+"\n");
    }

    public static void main (String args[]) {
        AddContacts myContacts= new AddContacts();
        myContacts.run();
        System.exit(0);
    }

    public void run() {
       // Construct a UDDIProxy object
       UDDIProxy testProxy = new UDDIProxy();

       try {
            testProxy.setInquiryURL(inquiryURL);
            testProxy.setPublishURL(publishURL);

            // Get an authorization token from the UBR
            print("** Get Authentication Token from the Registry **");
            AuthToken token = testProxy.get_authToken(USERID,PASSWD);
            print("Returned Token: "+ token.getAuthInfoString());
            print("Operator of the UBR:"+token.getOperator());

             //set up one contact for my business
            print("Set up of new Contact Information!");
            Contacts  konecoContacts = new Contacts();
            Vector MailVector = new Vector();
            Vector PhoneVector = new Vector();

            Phone konecoPhone = new Phone("+49 2247 4702");
            Email konecoMail = new Email("info@koneco.de");
```

```
            PersonName contactPerson = new PersonName("Reto Kortas");

            MailVector.add((Object)konecoMail);
            PhoneVector.add((Object)konecoPhone);

            Contact firstContact = new Contact();
            firstContact.setEmailVector(MailVector);
            firstContact.setPhoneVector(PhoneVector);
            firstContact.setUseType("sales contact + technical contact");
            firstContact.setPersonName(contactPerson);

            konecoContacts.add(firstContact);

            BusinessEntity be = new BusinessEntity(BKEY,businessName);
            //add the contact information to the existing business
            be.setContacts(konecoContacts);
            be.setDefaultDescriptionString("IT-Dienstleistungen aller Art");
            Vector entities = new Vector();
            entities.addElement(be);
            print("Save modified  businessEntity in the UBR!");
            BusinessDetail bd =
             testProxy.save_business(token.getAuthInfoString(),entities);

            // Process returned BusinessDetail object
            print("Process returned Informations!");
            Vector businessEntities = bd.getBusinessEntityVector();
            BusinessEntity returnedBusinessEntity =
             (BusinessEntity)(businessEntities.elementAt(0));
            print("Returned businessKey: " + returnedBusinessEntity.getBusinessKey());
            print("Returned Operator: "+returnedBusinessEntity.getOperator());
            print("New saved Contact Informations:");
            Contacts newContacts = returnedBusinessEntity.getContacts();
            Vector contactsVector = newContacts.getContactVector();
            Contact newFirstContact = (Contact)contactsVector.firstElement();
            print("Use contact for: "+newFirstContact.getUseType());
            print("Contact Pesron: "+newFirstContact.getPersonNameString());
            print("E-mail address:
             "+((Email)(newFirstContact.getEmailVector().firstElement())).getText());
            print("Phone number:
             "+((Phone)(newFirstContact.getPhoneVector().firstElement())).getText());

        }
        // Handle possible errors
        catch (UDDIException e) {
            DispositionReport dr = e.getDispositionReport();
            if (dr!=null) {
                System.out.println("UDDIException faultCode:" + e.getFaultCode() +
                                   "\n operator:" + dr.getOperator() +
                                   "\n generic:"  + dr.getGeneric() +
                                   "\n errno:"    + dr.getErrno() +
                                   "\n errCode:"  + dr.getErrCode() +
                                   "\n errInfoText:" + dr.getErrInfoText());
            }
            e.printStackTrace();

        // Catch any other exception that may occur
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

7.3 FindBusiness

Mit „FindBusiness.java“ kann man nach Unternehmen in einer Registry suchen. Der Namen, nach dem gesucht werden soll, oder auch nur ein Teil-Name, wird über die Kommandozeile angegeben. Auf die Anfrage „find_business“ erhält man ein „businessList“ Objekt, welches alle Unternehmen enthält, welche den angegebenen Parametern (businessName) entsprechen. Mit der Methode „processBusinessDetails” werden über den angegebenen „businessKey” alle Kontakt-Informationen zu dem Unternehmen aus der UBR ausgelesen.

```
import org.uddi4j.*;
import org.uddi4j.client.*;
import org.uddi4j.datatype.*;
import org.uddi4j.datatype.assertion.*;
import org.uddi4j.datatype.binding.*;
import org.uddi4j.datatype.business.*;
import org.uddi4j.datatype.service.*;
import org.uddi4j.datatype.tmodel.*;
import org.uddi4j.request.*;
import org.uddi4j.response.*;
import org.uddi4j.util.*;

import org.w3c.dom.Element;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.util.Vector;
import java.util.Properties;
import java.io.*;


public class FindBusiness {
     //addresses of the IBM TestRegistry
    String inquiryURL =
       "http://www-3.ibm.com/services/uddi/testregistry/inquiryapi";
    String publishURL =
       "https://www-3.ibm.com/services/uddi/testregistry/protect/publishapi";

    public static String businessName;

    public static void print(String messageToPrint){
        System.out.println(messageToPrint+"\n");
    }


    public static void processBusinessDetails(UDDIProxy proxy, String bk){
       try{
              BusinessDetail bd =  proxy.get_businessDetail(bk);
              print("Process returned Informations!");
              Vector businessEntities = bd.getBusinessEntityVector();
              BusinessEntity returnedBusinessEntity =
                     (BusinessEntity)(businessEntities.elementAt(0));
              print("Returned businessKey: " +
                     returnedBusinessEntity.getBusinessKey());
              print("Returned Operator: "+returnedBusinessEntity.getOperator());
              Contacts contacts = returnedBusinessEntity.getContacts();
              Vector contactsVector = contacts.getContactVector();
              Contact contact = (Contact)contactsVector.firstElement();
              if(contact.getUseType()!="")
              print("Use contact for: "+contact.getUseType());
              print("Contact Person: "+contact.getPersonNameString());
              if(contact.getEmailVector().isEmpty() == false)
              print("E-mail address:
                     "+((Email)(contact.getEmailVector().firstElement())).getText());
              if(contact.getPhoneVector().isEmpty() == false)
              print("Phone number:
```

```
                "+((Phone)(contact.getPhoneVector().firstElement())).getText())
    }
     catch (UDDIException e) {
           DispositionReport dr = e.getDispositionReport();
           if (dr!=null) {
               System.out.println("UDDIException faultCode:" + e.getFaultCode() +
                                  "\n operator:" + dr.getOperator() +
                                  "\n generic:"  + dr.getGeneric() +
                                  "\n errno:"    + dr.getErrno() +
                                  "\n errCode:"  + dr.getErrCode() +
                                  "\n errInfoText:" + dr.getErrInfoText());
           }
           e.printStackTrace();
    }
     catch (Exception e) {}
}

public static void main (String args[]) {
     FindBusiness myBusiness= new FindBusiness();
     businessName = args[0];
     myBusiness.run();
     System.exit(0);
}

public void run() {
      // set-up a new UDDIProxy object
    UDDIProxy testProxy = new UDDIProxy();

     try {
         testProxy.setInquiryURL(inquiryURL);
         testProxy.setPublishURL(publishURL);
          //creating vector of Name Object
         Vector names = new Vector();
         names.add(new Name(businessName));
         // Setting FindQualifiers to 'caseSensitiveMatch'
         print("Set up Find Qualifieres !");
         FindQualifiers findQualifiers = new FindQualifiers();
         Vector qualifier = new Vector();
         qualifier.add(new FindQualifier("caseSensitiveMatch"));
         findQualifiers.setFindQualifierVector(qualifier);
         // Find businesses by name and return max 5 businesses
         print("Searching for Businesses!");
         BusinessList businessList = testProxy.find_business(names, null, null,
          null,null,findQualifiers,5);
         print("Found Businesses:");
         Vector businessInfoVector  =
          businessList.getBusinessInfos().getBusinessInfoVector();
         for (int i = 0; i < businessInfoVector.size(); i++) {
          BusinessInfo businessInfo =
          (BusinessInfo)businessInfoVector.elementAt(i);

          print("************************************************************");
          print("Business: "+businessInfo.getNameString());
          print("Key: "+businessInfo.getBusinessKey());
          print("Description: "+businessInfo.getDefaultDescriptionString());
          print("Further Business Information:");
          processBusinessDetails(testProxy,businessInfo.getBusinessKey());
          print("************************************************************");
         }
     // catch UDDI Exceptions
     }
     catch (UDDIException e) {
         DispositionReport dr = e.getDispositionReport();
         if (dr!=null) {
             System.out.println("UDDIException faultCode:" + e.getFaultCode() +
                                "\n operator:" + dr.getOperator() +
                                "\n generic:"  + dr.getGeneric() +
```

```
                                    "\n errno:"    + dr.getErrno() +
                                    "\n errCode:"  + dr.getErrCode() +
                                    "\n errInfoText:" + dr.getErrInfoText());
                }
                e.printStackTrace();
            }
            // catch regular Exceptions
            catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```