Arbeitsgruppe Datenbanken und Informationssysteme
Universität Kaiserslautern

Seminar SS 2001:

DB-Aspekte des E-Commerce, Schwerpunkt: Techniken

UDDI – Universal Description, Discovery and Integration

von
Andreas Riethe
a_riethe@informatik.uni-kl.de

Inhaltsverzeichnis

1. Einleitung

Unternehmen benutzen das Internet heutzutage auf verschiedene Weise. Viele bedienen sich der vorhandenen Netzinfrastruktur und definieren spezifische Schnittstellen, damit ihre eigenen Anwendungen mit den Anwendungen ihrer Geschäftspartner zusammenarbeiten. Ohne die Übereinkunft für einen Standard bilden sich somit viele, jeweils einzigartige Strukuren, die zwar ihren eigens auferlegten Zweck erfüllen, aber auch einen hohen (Wartungs)aufwand mit sich bringen. Inzwischen hat sich schon eine ganze Industrie gebildet, die Integrationslösungen zwischen „inkompatiblen" Systemen anbietet.
Allerdings gibt es mittlerweile Anzeichen dafür, dass die Sprache „Extensible Markup Language" (XML) eine größere Rolle bei dem Datenaustausch zwischen verschiedenen Firmen spielt. Dies ist als vorteilhaft zu bewerten, da XML als gemeinsames Informationsformat, welches zum einen die Daten und zum anderen das Format in Form eines DTD (Data Type Definition) bereitstellt, nicht von der jeweiligen Implementierungssprache abhängt. Weiterhin sorgt ein Protokoll namens „Simple Object Access Protocol" (SOAP) dafür, dass die Interaktion zwischen verschiedenen Betriebssystemen aufgrund der Plattformunabhängigkeit stark vereinfacht wird. In der Funktionsweise ist dies etwa mit einem „Remote Method Invocation" (RMI) in Java vergleichbar. Insgesamt lässt sich derzeit ein Schichtenmodell bilden, bei dem die oberen Schichten die unteren benutzen und zur Hilfe nehmen, um ihre eigene Funktionalität zu erreichen.
Im Folgenden möchte ich einen kleinen Überblick über die derzeitige Schichtenstruktur geben. Als unterste Schicht werden die bekannten Internetprotokolle „HyperText Transfer Protocol" (HTTP) und „Transmission Control Protocol / Internet Protocol" (TCP/IP) verwendet. Diese Schicht sorgt also dafür, dass Anfragen registriert, Dateipakete an die richtige Stelle geschickt und in der richtigen Reihenfolge wieder zusammengesetzt werden. Für die zweite und dritte Schicht haben wir die bereits oben erwähnten Strukturen XML und SOAP. Als vierte Schicht steht dann das noch relativ neue Konzept des „Universal Description,Discovery and Integration" (UDDI), was auch zugleich das Thema dieser Ausarbeitung ist.
Aber zunächst noch einmal das Schichtenmodell im Überblick (Abb. 1):

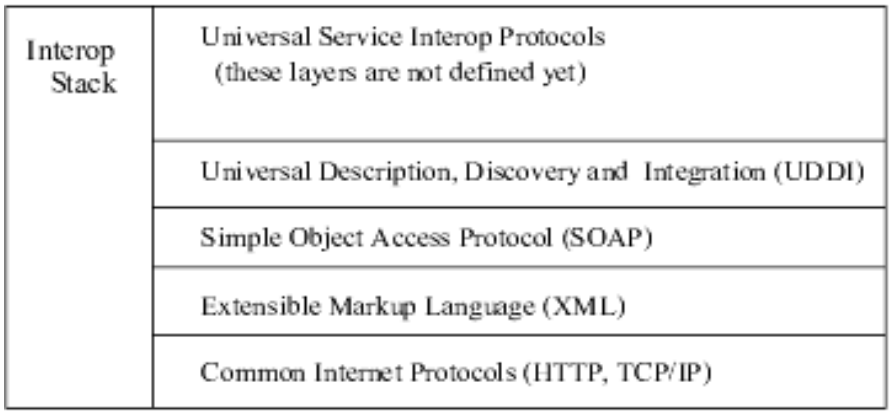

Abb. 1 Schichtenmodell; Ref. [1] UDDI Technical White paper (S. 3)

An diesem Schaubild kann man auch erkennen, daß UDDI noch nicht als das Ende der Fahnenstange angesehen wird. Vielmehr werden in Zukunft noch andere Protokolle auf UDDI aufsetzen und somit eine immer bessere und komfortablere Anfragebedienung ermöglichen.

1.1. Die Geschichte von UDDI

Als namhafte Gründungsväter von UDDI sind vor allem Ariba, IBM und Microsoft zu nennen. Nachdem in den letzten Jahren ein starkes Wachstum in dem Bereich des B2B eCommerce zu verzeichnen war, haben sich diese Firmen vorgenommen, gemeinsam eine Lösung zu den noch bestehenden Problemen zu finden. Die angesprochenen Probleme waren vor allem in der individuellen Ausrichtung der firmeneigenen eCommerce Gestaltung begründet. Dadurch gab es keine übergeordnete Stuktur und die Möglichkeiten Handel zu betreiben war nur auf lokal bekannte Firmen oder Branchen beschränkt. Außerdem kam unter denen, die in Frage kamen, noch das Problem der Inkompatibilität ihrer nicht selten verschiedenen Anwendungen und Webservices hinzu. Als mittelbare Folge der hohen Kosten für einzelne Integrationslösungen wurden kleinere Unternehmen von der Möglichkeit, am digitalen Markt teilzunehmen, ausgeschlossen.
Nach etwa sechs Monaten und 50 Meetings wurde dann die erste Spezifikation von UDDI am 30. September 2000 fertig gestellt.

1.2. Was ist UDDI

[Zitat: http://www-3.ibm.com/services/uddi/standard.html]
„The UDDI (Universal Description, Discovery and Integration) Project is a comprehensive, open industry initiative enabling businesses to (I) discover each other, and (II) define how they interact over the internet and (III) share information in a global registry architecture. UDDI is the building block which will enable businesses to quickly, easily and dynamically find and transact with one another via their preferred applications.“

Wie dieses Zitat schon sagt, zielt UDDI im Wesentlichen auf drei Eckpfeiler ab. Zum Einen die Möglichkeit potentielle Geschäftspartner zu finden, zum Anderen deren eingesetzte technische Systeme wegen eventuellen Inkompatibilitäten zu beschreiben und zum Dritten, eine Registrierungsdatenbank mit globaler und branchenübergreifender Reichweite anzulegen.

1.3. Vorteile von UDDI

Die Registrierung ist für Unternehmen jeder Größe geeignet, da in der Spezifikation besonders auf Probleme, die das Wachstum und Synenergien von B2B Handel beschränken, eingegangen wird. Außerdem ist die Nutzung des Service sowie die Registrierung über Ariba, IBM oder Microsoft zumindest vorläufig noch kostenlos. Dies hat sicherlich auch den Grund, möglichst viele Unternehmen dafür zu gewinnen, den Service in ihren Geschäftsalltag zu integrieren, denn ein solches Projekt lebt schließlich von seinen Teilnehmern. Die Tatsache, dass solche Marktgrößen wie Ariba, IBM und Microsoft an der Entwicklung beteiligt sind, gibt zusätzlich etwas Sicherheit, dass UDDI zu einem Standard und nicht im Sande verlaufen wird. Die ursprüngliche Anzahl von 36 Unternehmen, die zugesagt haben UDDI zu unterstützen, ist mittlerweile auf über 220 gestiegen. Weitere Vorteile sind, ein „passendes“ Unternehmen unter weltweit Millionen herauszufinden und die notwendigen technischen Informationen für eine Inanspruchnahme des jeweiligen Service zu erfahren. Außerdem bietet sich natürlich durch die Registrierung die Möglichkeit neue Kunden zu erreichen, bzw. erreicht zu werden. Dies bezieht sich sowohl auf neue Kundengruppen im Umland als auch durch die globale Präsenz auf Kundengruppen außerhalb der vorherigen Marktreichweite. Weiterhin kann man sein Angebot an Webservices ausbauen, da diese dank einer strukturierten Suchmöglichkeit auch gefunden werden. Ein anderer Vorteil ist zudem, dass sämtliche Unternehmen ihre Dienste und Geschäftsprozesse an einer einzigen, offenen und seriösen Stelle registrieren können, so daß unnötiges Suchen an anderen Stellen überflüssig wird.

2. Die Struktur von UDDI

Um die vorher schon angesprochene Komfortabilität bei Suche nach bestimmten Unternehmen oder Services zu erreichen, ist es notwendig, Informationen über die Unternehmen zu sammeln und strukturiert zu speichern. Dazu dient die sogenannte Business Registration. Hier können drei verschiedenartige, relevante Informationsangaben gemacht werden, die man oft in Analogie zu Telefonbüchern mit „White pages“, „Yellow pages“ und „Green pages“ bezeichnet. Die „White Pages“ enthalten Angaben wie den Namen des Unternehmens, eine textuelle Beschreibung (Liste von Beschreibungen in mehreren Sprachen), Kontaktinformation (wie Adresse, Telefonnummer, e-mail, URI, usw.) und einer Liste von Identifikatoren, unter denen das Unternehmen noch eindeutig bekannt ist (z. Bsp. DUNS). Die „Yellow pages“ enthalten Angaben darüber, in welche Kategorie das Unternehmen fällt. Bei der derzeitigen Version 1 von UDDI gibt es drei Taxonomien, d. h. Einordnungsaspekte, die das Unternehmen näher charakterisieren soll. Diese sind jeweils der Industriezweig (nach NAICS), das Produkt/Service (nach UN/SPSC) und die geografische Einordnung. Daneben gibt es eigentlich noch eine vierte, nämlich „Other“, in die alle sonstigen Kategorien fallen. Schließlich enthalten die „Green pages“ noch Information, die technische Angaben über die Art und Weise macht, wie man die Webservices des Unternehmens benutzen kann. Dazu zählen unter anderem die Servicebeschreibung, verwendete Plattform, verwendete Anwendungen, usw.. Die entsprechenden Informationen können von den Unternehmen entweder über eine Internetseite eingegeben werden, oder unter Verwendung von geeigneten Werkzeugen, die die UDDI API unterstützen. Außerdem hat man die Wahl, sich unter einem von mehreren Partnerservern zu registrieren. Diese werden täglich auf ihre Einträge hin abgestimmt, so dass fast zu jeder Zeit die gleichen Einträge auf jedem Partnerserver vorhanden sind, das sogenannte „registered once, published everywhere“ Prinzip.
Daneben gibt es noch eine „Service Type Registration“, die von Standardisierungsgesellschaften, Programmierern, aber auch Unternehmen durchgeführt werden. Diese Registrierung enthält Angaben über verschiedene Servicetypen, eine Referenz, die auf eine Beschreibung des jeweiligen Typs verweist, Informationen darüber, wer den Servicetyp veröffentlicht hat, und eine eindeutige Identifikation für den jeweiligen Servicetyp, die auch als „tModelKey“ bezeichnet wird. Diese Identifikation wird auch von den Internetseiten benutzt, die diesen Service verwenden. Insgesamt lässt sich somit die Information in der „Service Type Registration“ mit Metadaten vergleichen.
Die folgende Abbildung 2 veranschaulicht nochmal die Funktionsweise von UDDI.

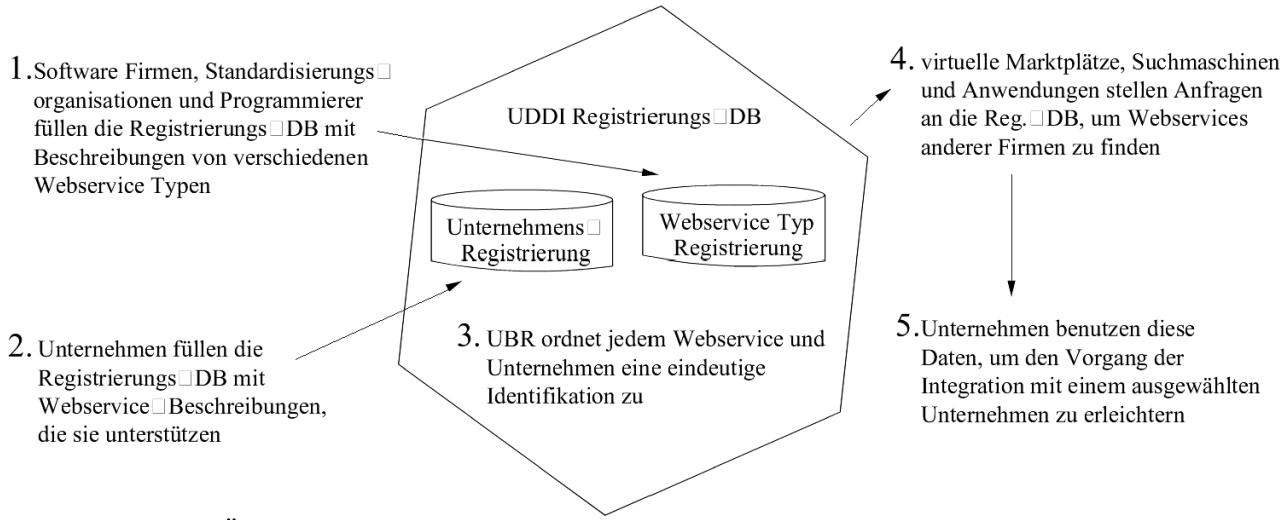

Abb. 2: UDDI Überblick; Ref. [3] UDDI Overview Presentation (Folie 7)

2.1. Übersichtsdiagramm

Aus technischer Sicht besteht die UDDI Spezifikation aus einem XML-Schema für SOAP Nachrichten und einer Beschreibung der UDDI API Spezifikation. Beides zusammen bildet die Basis für ein Informationsmodell und Interaktionsframework, das es ermöglicht, Informationen über Webservices zu suchen und zu veröffentlichen. Insgesamt gibt es etwa 30 SOAP Nachrichten, die Anfrage- und Veröffentlichungsfunktionen an die UDDI Business Registry stellen. Die Entscheidung für den Einsatz von XML liegt vor allem in der plattformunabhängigen Sicht auf die Daten und der Möglichkeit, hierarchische Beziehungen in einer natürlichen Weise darzustellen. Das XML-Schema definiert vier Kerntypen, die die für die Nutzung von Webservices notwendigen Informationen bereitstellen. Diese sind: businessEntity, businessService, bindingTemplate und tModel. Bei der ersten Veröffentlichung eines „neuen" Unternehmens werden diesen XML-Elementen jeweils eindeutige Identifikationen zugeteilt, da auf die einzelnen Elemente später auch unabhängig voneinander zugegriffen werden kann. Diese Identifikation wird als Universal Unique ID (UUID) bezeichnet.
Die folgende Abb. 3 zeigt eine Übersicht der verschiedenen Typen mit jeweiligen Attributen und Beziehungen untereinander:

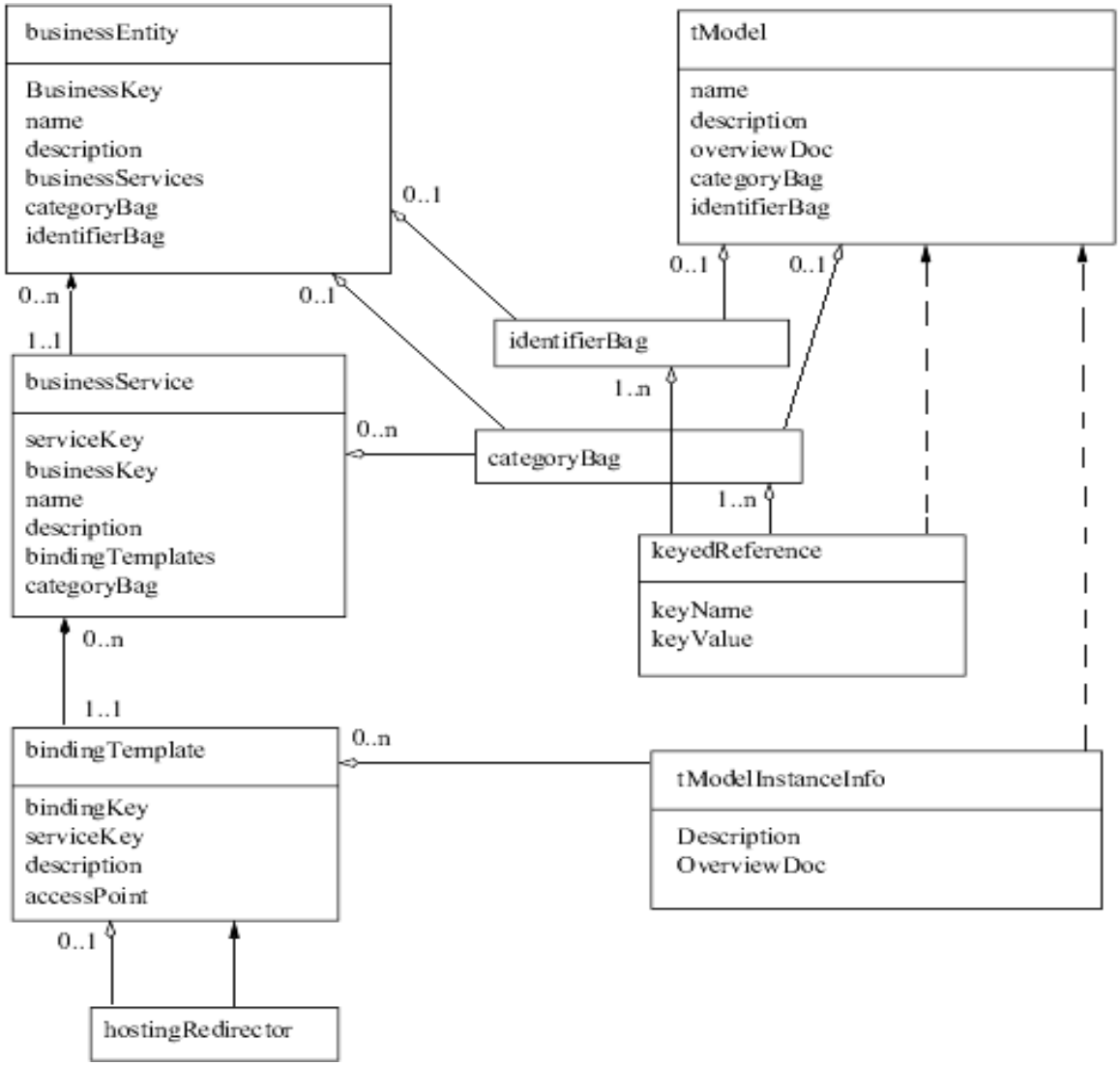

Abb. 3: Informationsmodell; Ref. [1] Technical White Paper (S. 11)

Auffällig ist hierbei die hierarchische Struktur. Die businessEntity-Klasse enthält eine oder mehr unterscheidbare businessService-Klassen und diese wiederum unterscheidbare bindingTemplate-Klassen. Die bindingTemplate-Klasse enthält unter anderem eine Referenz auf eine spezielle tModel-Klasse. Wichtig ist, daß eine Instanz der vier Strukturen niemals in mehr als einer Vaterstruktur enthalten ist. Das heißt also zum Beispiel, dass nur genau eine businessEntity-Klasse Informationen zu einer bestimmten businessService-Klasse bereitstellt. Anders ist dies bei Referenzen, wie man am Beispiel der bindingTemplate-Klasse sehen kann. Hier können mehrere verschiedene bindingTemplate-Klassen auf dieselbe tModel-Klasse verweisen.
Daneben gibt es noch die Klasse „identifierBag“, die es businessEntity und tModel erlaubt, zusätzliche eindeutige Formen der Identifikation als Name-Wert Paar in der Klasse „keyedReference“ zu speichern. Weiterhin erlaubt die Klasse „categoryBag“ den Klassen „businessEntity“, „businessService“ und „tModel“ eine Einordnung ihrer Daten in folgende drei verschiedenene Taxonomien: Industriezweig, Service/Produkt und geographische Region. Diese Information wird dann ebenfalls als Name-Wert Paar in der Klasse „keyedReference“ gespeichert. Die Kategorisierung ist von zentraler Bedeutung für UDDI, weil es ohne sie sehr schwer und ineffizient wäre, bestimmte Daten zu finden. Andererseits darf man nicht annehmen, dass die UDDI Datenbank gut geeignet für allgemeine Firmensuchen ist. Das liegt daran, dass man bald von einer Größe von mehreren hunderttausend bis einer Millionen verschiedenen registrierten Unternehmen ausgeht. Das Ergebnis einer Suchanfrage, selbst mit stark einschränkenden Kriterien, wäre wahrscheinlich immer noch viel zu groß und würde nicht sonderlich weiterhelfen. Ein zweites Problem ist, dass UDDI nur nach Begriffen mit exakter Übereinstimmung suchen kann. So würde etwa eine Suchanfrage in der Kategorie „Handel-Gemüse“ und „Handel“ zwei disjunkte Ergebnismengen zurückliefern. Deshalb werden für Anfragen dieser Art intelligente Suchmaschinen benötigt, die schon über ein a priori Wissen für bestimmte Kategorien und deren Zusammenhänge haben. Die Fähigkeit Unternehmen zu finden, die einen bestimmten Service zu einem bestimmten Preis innerhalb eines bestimmten Gebietes bis zu einem bestimmten Datum bieten können, ist nicht direkt mit UDDI möglich. Vielmehr bietet UDDI das Informationsmodell, um Anfragen dieser Art in höheren Schichten stellen zu können. Das Design von UDDI erlaubt nur eher einfache Formen der Suche. Allerdings können die Unternehmen beim Registrieren neben den Daten über sich selbst und ihre Webservices auch zusätzliche Kategorisierungsangaben machen, die von intelligenten Suchmaschinen benutzt werden.
Die folgende Abb. 4 stellt diesen Sachverhalt noch einmal graphisch da:

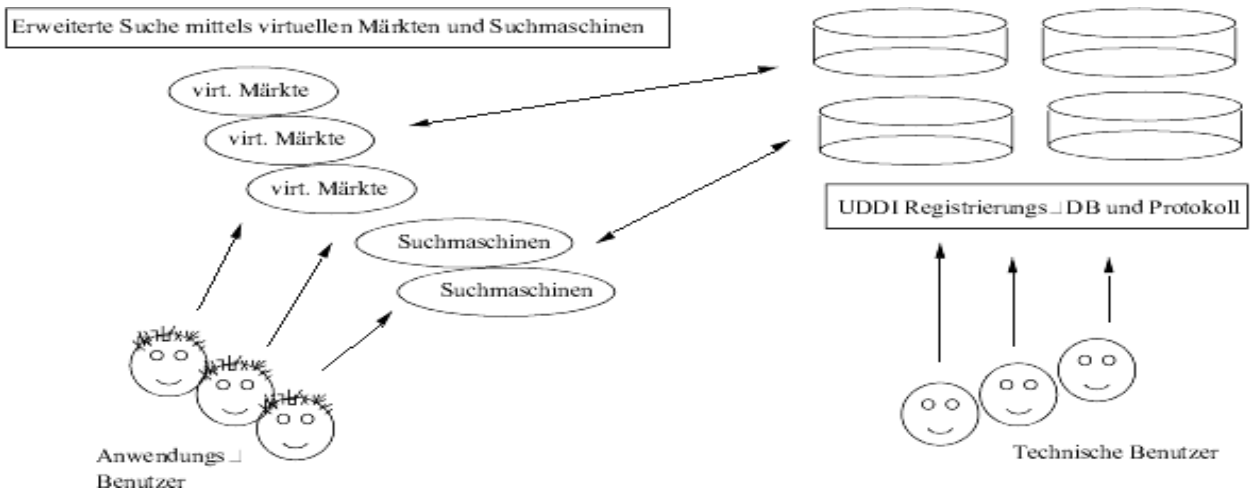

Abb. 4 UDDI Suchkonzept; Ref. [2] UDDI Data Structure Reference V1.0 (S.24)

2.2. Die Klasse businessEntity

Diese Klasse enthält sozusagen die „White pages“ Informationen über eine jeweilige Geschäftseinheit. Aus der hierarchisch organisierten XML-Sicht handelt es sich hierbei um die oberste Datenstruktur, die alle anderen Substrukturen beinhaltet. Typischerweise bietet diese Klasse den Einstiegspunkt einer Suche, wenn man sich über die Dienste eines bestimmten Unternehmens informieren möchte. Hilfreich ist auch, dass hierbei eine Suche gemäß den bereits erwähnten drei Taxonomien der „Yellow pages“ unterstützt wird. Damit ist es also möglich, innerhalb eines bestimmten Industriezweiges, einer Produktkategorie, oder einer gewählten geographischen Region zu suchen.
Die XML-Definition einer businessEntity sieht folgendermaßen aus:

```
<element name = "businessEntity">
        <type content = "elementOnly">
                <annotation>
                        <appInfo>
        Primary Data type: Describes an instance of a business or business unit.
                        </appInfo>
                </annotation>
                <group order = "seq">
                        <element ref = "discoveryURLs" minOccurs = "0" maxOccurs = "1"/>
                        <element ref = "name"/>
                        <element ref = "description" minOccurs = "0" maxOccurs = "*"/>
                        <element ref = "contacts" minOccurs = "0" maxOccurs = "1"/>
                        <element ref = "businessServices" minOccurs = "0" maxOccurs = "1"/>
                        <element ref = "identifierBag" minOccurs = "0" maxOccurs = "1"/>
                        <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1"/>
                </group>
                <attribute name = "businessKey" minOccurs = "1" type = "string"/>
                <attribute name = "operator" type = "string"/>
                <attribute name = "authorizedName" type = "string"/>
        </type>
</element>
```

Quelle: [5] UDDI XML Schema

Wie man sieht, gibt es bei einer businessEntity drei Attribute. Davon ist der „businessKey“ besonders wichtig, da dieser den schon zuvor erwähnten Universal Unique ID (UUID) enthält. Damit ist also eine Instanz der businessEntity eindeutig definiert. Weitere Attribute sind „operator“ und „authorizedName“. Ersteres enthält den Namen des Betreibers der zuständigen UDDI Registrierungsseite. Letzteres enthält den Namen der Person, die die businessEntity veröffentlicht. Beide Angaben werden automatisch vom System ausgefüllt. Bei den Elementen kann man zwischen einfachen und zusammengesetzten unterscheiden. Einfache Elemente sind in diesem Fall „name“ und „description“. Für „name“ ist es als einziges Element notwendig eine Angabe zu machen, da dieses den Namen des Unternehmens widerspiegelt und dieser für eine spätere Suche unerlässlich ist. Alle anderen Elemente können optional angegeben werden. „description“ enthält eine Beschreibung des jeweiligen Unternehmens, wobei jeweils nur eine Beschreibung, allerdings in mehreren Sprachen erlaubt ist. „discoveryURLs“ ist eine Substruktur, die eine Liste von URLs bereitstellt, welche auf URL-adressierbare Dokumente mit dem Mechanismus HTTP-GET zugreift. Weiterhin bietet die Substruktur „contacts“ Informationen über Name, Telefonnummer, Adresse, usw.. „businessServices“ besteht nur aus einer Sammlung von businessService Strukturen. „identifierBag“ enthält zusätzliche Identifikatoren, unter denen das jeweilige Unternehmen auch noch eindeutig bestimmt werden kann. Hierzu zählt zum Beispiel eine Steuernummer oder eine Dun&Bradstreet DUNS Nummer. Die Angabe solcher zusätzlichen Identifikationen ist optional, kann sich aber bei einer Suche positiv auswirken. Unter dem Element „categoryBag“ befinden sich

Angaben zu den schon zuvor eingeführten drei Taxonomien.

2.3. Die Klasse businessService

Diese Klasse dient dazu, um ähnliche Webservices in Gruppen zusammenzufassen. Eine Ähnlichkeit besteht, falls die Webservices zu einem gemeinsamen Geschäftsprozess gehören, oder falls sie in der gleichen Kategorie einzuordnen sind. Beispiele für die Ähnlichkeit bezüglich eines Geschäftsprozesses sind etwa die Gruppe der Bezahlservices oder Versandservices, etc..

Die XML-Definition eines businessService sieht folgendermaßen aus:

```
<element name = "businessService">
        <type content = "elementOnly">
                <annotation>
                        <appInfo>
        Primary Data type: Describes a logical service type in business terms.
                        </appInfo>
                </annotation>
                <group order = "seq">
                        <element ref = "name"/>
                        <element ref = "description" minOccurs = "0" maxOccurs = "*"/>
                        <element ref = "bindingTemplates"/>
                        <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1"/>
                </group>
                <attribute name = "serviceKey" minOccurs = "1" type = "string"/>
                <attribute name = "businessKey" type = "string"/>
        </type>
</element>
```

Quelle: [5] UDDI XML Schema

Das Attribut „businessKey“ enthält die UUID der zugehörigen businessEntity. Falls businessService in einer kompletten businessEntity enthalten ist, die bereits ihren businessKey liefert, ist die Angabe optional.Werden die Daten von businessService aber gesondert betrachtet, ohne den Bezug einer businessEntity, ist die Angabe notwendig. Damit wird die Möglichkeit zugesichert, zwischen der Vater-Sohn Relation hin- und herzuwechseln, unabhängig davon, ob man bei businessEntity oder businessService anfängt. Das andere Attribut „serviceKey“ ist die UUID für businessService und damit auch notwendig. Beim Abspeichern eines neuen businessService übergibt man dieses Attribut ohne Wert, woraufhin systemseitig ein UUID Wert generiert und zugewiesen wird. Das Element „name“ ist notwendig und enthält einen Namen für die jeweilige Serviceklasse in menschenverständlicher Form. Das Element „description“ ist optional und kann null bis mehrere Beschreibungen für die jeweilige Serviceklasse enthalten. „bindingTemplates“ ist ein Container für mehrere bindingTemplate Strukturen und enthält technische Beschreibungen zu den entsprechenden Dienstklassen. Unter dem Element „categoryBag“ befinden sich Angaben zu den schon zuvor eingeführten drei Taxonomien, die zur Verbesserung von Suchanfragen beitragen können.

2.4. Die Klasse bindingTemplate

Diese Struktur enthält alle Informationen, die für ein Anwendungsprogramm relevant sind, um mit einem anderen Webservice zu kommunizieren. Diese Informationen beinhalten sowohl die Internetadresse eines Webservices, als auch eventuelle Voraussetzungen für dessen Nutzung, wie zum Beispiel die Installation bestimmter Softwarepakete. Weiterhin ist ein Mechanismus enthalten,

der es erlaubt komplexe Routing-Optionen, wie etwa Lastbalancierung, durchzusetzen.

Die XML-Definition eines bindingTemplate sieht folgendermaßen aus:

```
<element name = "bindingTemplate">
        <type content = "elementOnly">
                <annotation>
                        <appInfo>
        Primary Data type: Describes an instance of a web service in technical terms.
                        </appInfo>
                </annotation>
                <group order = "seq">
                        <element ref = "description" minOccurs = "0" maxOccurs = "*"/>
                        <group order = "choice">
                                <element ref = "accessPoint" minOccurs = "0" maxOccurs = "1"/>
                                <element ref = "hostingRedirector" minOccurs = "0" maxOccurs = "1"/>
                        </group>
                        <element ref = "tModelInstanceDetails"/>
                </group>
                <attribute name = "bindingKey" minOccurs = "1" type = "string"/>
                <attribute name = "serviceKey" type = "string"/>
        </type>
</element>
```

Quelle: [5] UDDI XML Schema

Das Attribut „bindingKey“ ist die UUID für die bindingTemplate Struktur. Wie zuvor beim serviceKey gilt auch hier ein leeres Feld beim Abspeichern als Aufforderung an das System, eine UUID zu generieren und zuzuweisen. Um eine bereits existierende bindingTemplate Instanz zu ändern, muss man die UUID dieser Instanz übergeben. „serviceKey“ ist die UUID zu einem bestimmten businessService und dient hier als Fremdschlüssel. Das Element „description“ ist optional und kann null bis mehrere technische Beschreibungen von einem bestimmten Service enthalten. „accessPoint“ ist ein Element, das immer über ein Name-Wert-Paar angegeben ist. Zum Beispiel kann als Name: phone und Wert: 0631- 110 angegeben sein. Allerdings enthält „accessPoint“ nur den Wert-Teil des Name-Wert-Paares. Ohne die technischen Voraussetzungen des Services und damit den zugehörigen Name-Teil zu kennen, kann man keine Aussage über den Inhalt von „accessPoint“ machen. Als mögliche Name-Teile stehen folgende zur Verfügung: mailto, http, https, ftp, fax, phone und other. Die jeweiligen Name-Teile implizieren, in welchem Format der Inhalt von „accessPoint“ ist. Das Element „hostingRedirector“ ist alternativ zu verwenden, falls „accessPoint“ nicht zur Verfügung steht. Es enthält eine Referenz zu einem anderen bindingTemplate, das bereits den jeweiligen Service beschreibt. Diese Vorgehensweise ist vor allem dann üblich, wenn eine Service-Beschreibung auf viele Services anwendbar ist. Das Element „tModelInstanceDetails“ ist eine Liste von tModelInfos. Alle Information zusammengenommen sollte einen eindeutigen „Fingerabdruck“ liefern, der es ermöglicht kompatible Services zu erkennen.

2.5. Die Klasse tModel

Die Klasse tModel enthält Metadateninformationen , die Webservices auf einer abstrakten Ebene beschreibt und eindeutig mit einem tModelKey bezeichnet. Dadurch stellt das tModel eine Möglichkeit bereit, gezielt nach kompatiblen Webservices zu suchen, indem man nach denjenigen Webservices mit zum eigenen System passendem tModelKey sucht. Der tModelKey wird deshalb auch als technischer Fingerabdruck bezeichnet. Webservices ihrerseits, die ein bestimmtes tModel

implementieren, geben eine Referenz davon über den eindeutigen tModelKey an.
Die zweite Anwendung von tModel Referenzen ist innerhalb von den Elementen „identifierBag“ und „categoryBag“. In diesem Zusammenhang stellt das tModel eine Beziehung zwischen eindeutigen Name-Wert-Paaren und ihrem übergeordneten Namen dar. Zum Beispiel besteht ein solches Name-Wert-Paar für „identifierBag“ etwa aus Steuernummer als Name und 012345 als Wert. Der übergeordnete Name ist dann etwa der Name eines eindeutig bestimmten Unternehmens. Weitere Beispiele für dise Anwendung sind die eingesetzten drei Taxonomien.

Die XML-Definition des tModel sieht folgendermaßen aus:

```
<element name = "tModel">
        <type content = "elementOnly">
                <annotation>
                        <appInfo>
                This structure defines a metadata about a technology, specification
                or namespace qualified list (e.g. taxonomy, organizaton, etc.)
                        </appInfo>
                </annotation>
                <group order = "seq">
                        <element ref = "name"/>
                        <element ref = "description" minOccurs = "0" maxOccurs = "*"/>
                        <element ref = "overviewDoc" minOccurs = "0" maxOccurs = "1"/>
                        <element ref = "identifierBag" minOccurs = "0" maxOccurs = "1"/>
                        <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1"/>
                </group>
                <attribute name = "tModelKey" minOccurs = "1" type = "string"/>
                <attribute name = "operator" type = "string"/>
                <attribute name = "authorizedName" type = "string"/>
        </type>
</element>
```
Quelle: [5] UDDI XML Schema

Das Attribut „tModelKey“ ist die UUID der tModel-Struktur und muss immer vorhanden sein. Wie zuvor wird auch hier beim Abspeichern einer neuen tModel-Struktur kein tModelKey-Wert übergeben, damit dieser dann systemseitig generiert und zugewiesen wird. Auch hier ist analog zu den bisherigen Strukturen im Falle eine Änderung an einem bereits existierenden tModel der entsprechende tModelKey mit anzugeben. Das Attribut „authorizedName“ enthält den Namen der Person, die die tModel-Daten veröffentlicht hat. Dieser Name wird ebenfalls von System generiert. Das ebenso systemgenerierte Attribut „operator“ enthält den Namen des Betreibers der zuständigen UDDI Registrierungsseite. Das Element „name“ beinhaltet den Namen für das entsprechende tModel und muss in jedem Fall angegeben werden, da er bei einer Suchanfrage als Kriterium angegeben wird. Des Weiteren sollte die Namensgebung intuitiv und passend zum jeweiligen tModel gewählt werden. Das Element „description“ ist optional und enthält eine kurze Beschreibung des tModels. Es ist jeweils nur eine Beschreibung pro Sprache erlaubt. Das Element „overviewDoc“ enthält zusätzliche Beschreibungen über die richtige Benutzung des tModels und Angaben über Internetadressen, die tiefergehende Informationen bereitstellen. „identifierBag“ und „categoryBag“ werden analog wie in der Klasse businessEntity verwendet.

3. API Referenz

Das Application Programming Interface (API) von UDDI ist so gestaltet, daß ein einfacher Anfrage/Antwort Mechanismus die Suche nach Unternehmen, Webservices oder technischen Informationen gestattet. Mit fortschreitender Zeit wird die API einigen Änderungen unterliegen. Deshalb ist es notwendig ihr eine Versionsnummer zuzuteilen, um zukünftige Versionen unterscheiden zu können und eine eindeutige Benutzungsrichtlinie zu bewahren. Da die API selbst

auf XML-Nachrichten beruht, wird die Versionsnummer in einem XML Attribut gespeichert. Generell unterscheidet man in der API zwischen zwei Typen von Methodenaufrufen. Das sind zum Einen die „Inquiry API functions“, die es ermöglichen Suchanfragen zu stellen und zum Anderen „Publishing API functions“, die zur Wartung und Neueinrichtung von eigenen Einträgen in der UDDI Registrierung dienen. Letztere erfordern allerdings zusätzlich noch einen Autorisierungsmechanismus, damit auch wirklich nur berechtigte Personen etwaige Änderungen vornehmen können. Letztlich ist auch noch eine Fehlerbehandlung beschrieben, die im Falle eines Fehlers aufschlussreiche Fehlerkodes zurückliefern soll. Viele dieser Aspekte werden im Folgenden näher betrachtet.

3.1. Inquiry API Funktionen

„Inquiry API functions“ stellen Methoden dar, die mit Hilfe eines SOAP Clients aufgerufen werden und Auskunft über Einträge in der UDDI Registrierung zurückliefern sollen. Alle Methodenaufrufe sind synchron, das heiß,t der Aufrufer befindet sich für die Dauer des Aufrufs in einer aktiven Warteschleife. Die Aufrufe können von jeder Person zu jeder Zeit an eine der UDDI Betreiberseiten gestellt werden. Insgesamt gibt es neun solcher Methodenaufrufe, die sich prinzipiell auf die vier Kerntypen von UDDI beziehen. Diese unterteilen sich in vier „find_*“ und fünf „get_*“ Methoden, die nachfolgend nur kurz aufgelistet werden sollen. Der interessierte Leser möge sich in der UDDI Programmer´s API (Ref. [4]) genauer informieren.

find_binding	get_bindingDetail
find_business	get_businessDetail; get_businessDetailExt
find_service	get_serviceDetail
find_tModel	get_tModelDetail

Allen Methoden gemein ist, dass sie jeweils eine bestimmte Auswahl an Parametern annehmen können, die über die Syntax festgelegt ist. Sollte ein Methodenaufruf eine zu große Treffermenge haben, wird der Rückgabewert auf eine erlaubte Menge reduziert, und ein dafür vorgesehenes „truncate“ Attribut auf true gesetzt. Falls ein Fehler während der Anfrageverarbeitung auftritt, wird eine entsprechende Nachricht an den Aufrufer gesendet.

3.1.1.Inquiry Patterns

Durch die API von Suchanfragen werden drei Muster, das heißt, abstrakte Suchverhalten, unterstützt. Das erste Pattern ist das sogenannte „browse pattern“. Dies beschreibt die Möglichkeit, mit irgendeiner gegebenen Information eine Suche zu starten. In UDDI wird dieses Pattern durch die find_* Methoden verwirklicht. Zum Beispiel kann man, um herauszufinden, ob ein bestimmtes Unternehmen bei UDDI registriert ist, eine find_business Methode benutzen. Damit hat man eine gute Möglichkeit für den Einstiegspunkt einer Suche. Möchte man allerdings dann doch mehr über ein bestimmtes Unternehmen wissen, ist ein weiteres Pattern notwendig. Dieses ist als sogenanntes „drill-down“ Pattern bekannt und wird bei UDDI durch die get_* Methoden verwirklicht. Typischerweise verwendet man die durch das „browse pattern“ gewonnene Information über eine UUID und setzt diese als Parameter in einer get_* Methode ein. Dadurch erhält man dann Zugriff auf alle Details eines bestimmten Unternehmens.
Das dritte Pattern, „invocation pattern“, bezieht sich auf die Nutzung von bindingTemplate Informationen, um einen fernen Webservice zu nutzen. Üblicherweise wird diese Information auf

dem lokalen System zwischengespeichert. Sollte sich diese Information ändern, etwa durch Server-Upgrades, Katastrophen-Recovery oder einfach nur durch Namensänderungen, wird der Aufruf dieses Webservices nicht mehr erfolgreich sein. In diesem Fall sieht das „invocation pattern“ vor, anstatt der zwischengespeicherten Information ein vielleicht aktuelleres bindingTemplate von der UDDI Registrierung zu holen. Danach wird vesucht, den Webservice mit der neuen Information aufzurufen. Sollte der Aufruf erfolgreich sein, wird die alte Information von der neuen ersetzt. Dieses Pattern kann dabei helfen, hohe Kosten für Kommunikations- und Koordinationsaufwand einzusparen, in dem man bei einer Änderung einfach die bindingTemplate Information aktualisiert, wodurch alle Systeme, die dieses Pattern unterstützen automatisch angepasst werden.

3.1.2. Search Qualifiers

Die Methoden find_binding, find_business, find_service und find_tModel erlauben jeweils noch ein optionales Element namens findQualifiers. Dieses Element überschreibt das standardmäßige Suchverhalten und bietet dafür andere an.
Die allgemeine Form des findQualifiers Elements sieht folgendermassen aus:

```
<findQualifiers>
        <findQualifier>fixedQualifierValue</findQualifier>
        [<findQualifier>fixedQualifierValue</findQualifier>  ...]
</findQualifiers>
```

[Ref. [4] UDDI Programmer´s API 1.0 (S.53)]

Derzeit sind offiziell sechs Werte für fixedQualifierValue definiert, die auf jeden Fall unterstützt werden. Einzelne Betreiberseiten bieten darüberhinaus noch weitere an. Die sechs fixedQualifierValues haben unterschiedliche Prioritäten und schließen sich teilweise gegenseitig aus. So haben „exactNameMatch“ und „caseSensitivNameMatch“ die höchste Priorität und können sogar kombiniert verwendet werden. Dabei wird dann das schwächere Kriterium die Ergebnismenge bestimmen. „sortByNameAsc“ und „sortByNameDesc“ haben die zweithöchste Priorität und schließen sich gegenseitig aus. Die niedrigste Priorität haben „sortByDateAsc“ und „sortByDateDesc“, welche sich ebenfalls gegenseitig ausschließen.

3.2. Publishing API Funktionen

„Publishing API functions“ stellen alle Methoden dar, die es ermöglichen, neue Einträge in der UDDI Registrierung zu machen, oder bestehende Einträge zu verändern. Damit jeder auch nur Zugriff auf die eigenen Einträge hat, ist bei einem Verbindungsaufbau eine Autorisierung nötig. Die Autorisierungsverfahren sind nicht in der UDDI API festgelegt, sondern werden von der jeweiligen UDDI Betreiberseite bestimmt. Sobald man sich einmal für eine Betreiberseite entschieden hat, ist es nur noch möglich über diese Eintragsänderungen vorzunehmen. Da es sich hierbei um sicherheitsrelevante Daten handelt, werden sämtliche Methodenaufrufe über HTTPS, eine Variante von HTTP mit Verschlüsselung, abgewickelt. Die insgesamt elf Methoden beziehen sich auch hier wieder auf die vier Kerntypen von UDDI, allerdings erweitert um zwei Methoden zur Autorisierung und einer Übersicht zur Bestimmung aller derzeitig veröffentlichten eigenen Einträge. Die elf Methoden sind im folgenden kurz aufgelistet. Auch hier wird der interessierte Leser gebeten, sich in der UDDI Programmer´s API (Ref. [4]) genauer zu informieren.

delete_binding	save_binding
delete_business	save_business
delete_service	save_service
delete_tModel	save_tModel
discard_authToken	get_authToken
	get_register

Die jeweiligen möglichen Parameter sind auch hier wieder durch die Syntax festgelegt. Bei erfolgreicher Durchführung wird für alle delete_* und die discard_AuthToken Methode ein sogenannter „dispositionReport“ zurückgeliefert, der einen Erfolgsindikator enthält. Sollte ein Fehler auftreten, wird dies ebenfalls in einem dispositionReport mit entsprechender Fehlermeldung zurückgeliefert. Für alle save_* Methoden wird bei erfolgreicher Durchführung eine Nachricht mit dem endgültigen Resultat dieses Aufrufs zurückgeliefert. „getAuthToken“ liefert eine authToken Nachricht zurück und „get_registeredInfo“ eine registeredInfo Struktur. Letzteres ist recht nützlich, um einen Überblick über die eigenen UDDI Einträge mit einem einzigen Methodenaufruf zu bekommen. In allen Fällen wird bei einem Fehler ein entsprechender dispositionReport zurückgeliefert.

3.3. Fehlerbehandlung

Alle Methodenaufrufe zu einer UDDI Betreiberseite werden mit Hilfe von SOAP durchgeführt. Danach wird ein dispositionReport an den Aufrufer zurückgegeben. Falls die Durchführung der Methode fehlerfrei war, wird ein dispositionReport mit Erfolgsmeldung zurückgeliefert, ansonsten einer mit entsprechender Fehlermeldung. Im Fehlerfall wird der dispositionReport in eine SOAP FAULT Struktur eingebettet. SOAP FAULT kennt drei Fehlerklassen: VersionMismatch, MustUnderstand und Client. Da in diesem Fall die Fehlerbehandlung mit einem dispositionReport eigentlich durch den Client erfolgt, wird eben diese Fehlerklasse gewählt. Es ist auch möglich, mehrere Fehler in einem dispositionReport zu sammeln. Dazu müssen mehrere result Elemente in demselben dispositionReport enthalten sein. Die Anzahl der möglichen result Elemente hängt von der jeweiligen Implementierung ab.
Im Folgenden ist die allgemeine XML-Struktur sowohl für den fehlerfreien als auch für den Fehlerfall dargestellt.

Eine mögliche Form eines dispositionReport im fehlerfreien Fall sieht so aus:[Ref. [4] S.45]

```
<?xml version="1.0" encoding="UTF-8"  ?>

<Envelope xmlns="http://schemas.xmlsoaporg.org/soap/envelope/">
<Body>
   <dispositionReport generic="1.0" operator="OperatorUrl" xmlns="urn:uddi-org:api">
      <result errno="0">
         <errInfo errCode="E_success" />
      </result>
   </dispositionReport>
</Body>
</Envelope>
```

Eine mögliche Form eines dispositionReport im Fehlerfall sieht so aus:[Ref. [4] S.45/46]

```
<?xml version="1.0" encoding="UTF-8"  ?>

<Envelope xmlns="http://schemas.xmlsoaporg.org/soap/envelope/">
<Body>
   <Fault>
      <faultcode>Client</faultcode>
      <faultstring>Client</faultstring>
      <detail>
         <dispositionReport generic="1.0" operator="OperatorUrl" xmlns="urn:uddi-org:api">
            <result errno="10050">
               <errInfo errCode="E_notSupported" >
                 The findQualifier value passed is unrecognized.
               </errInfo>
            </result>
         </dispositionReport>
      </detail>
   </Fault>
</Body>
</Envelope>
```

4. Ausblick

Das noch relativ junge Projekt UDDI erfreut sich einem regen Zuspruch in der Industrie. Sei es durch die einflussreichen Firmen, die hinter UDDI stehen, der geschickte Aufsatz auf die bereits existierenden und verwendeten Elemente XML und SOAP oder weil es ganz einfach der richtige Zeitpunkt für eine Weiterentwicklung in dieser Richtung ist. Wahrscheinlich ist es von Allem etwas.

4.1. Roadmap

Abb. 5 UDDI Roadmap [Ref. [3] UDDI Overview Presentation (Folie 20)

Obige Abb. 5 zeigt die geplante Entwicklung von UDDI bis Ende des Jahres 2001.
Bevor UDDI zu einem neuen Standard erklärt werden soll, sind noch zwei weitere Versionen geplant, die über eine erhöhte Funktionalität verfügen sollen. Im Bereich der „White page“ Daten soll dann statt der bisherigen Möglichkeit einzelne Unternehmen zu registrieren ganze Korporationen und schließlich jegliche Geschäftsbeziehungen unter verschiedenen Unternehmen registriert werden können. Im Bereich der „Yellow page“-Daten ist geplant, die Anzahl der bisher drei Taxonomien zu erhöhen und in Version 3 sogar auf einzelne Bedürfnisse derAnwender anzupassen. Das Ziel für die „Green page“-Daten ist es von der bisherigen Beschreibung einzelner Dienste über mehrschichtige Dienste bis hin zu der Beschreibung von kompletten Workflows zu kommen. Nachdem Version 3 erreicht sein wird, ist noch eine Fortführung der Entwicklung von UDDI geplant.

4.2. Meinung anderer Firmen

Die Meinung anderer Firmen über das UDDI Projekt ist generell als positiv anzusehen. Die meisten Äußerungen im Internet zu UDDI sind von Unternehmen, die selbst registrierte Mitglieder sind. Von daher ist diese Vermutung nicht verwunderlich. Inzwischen ist die Anzahl der registrierten Unternehmen auf über 220 angestiegen. Da sich vor allem auch große Unternehmen bereits registriert haben, ist UDDI auf dem besten Weg, sich als Standard durchzusetzen.
Abschliessend möchte ich noch ein paar selektive Stellungnahmen verschiedener Unternehmen angeben. Quelle: [15] Presse Artikel: UDDI : A New Proposed Standard...

Ariba:

"B2B eCommerce has seen rapid global adoption due to its ability to deliver unprecedented fast time to value, but this success has been uneven as marketplaces, buyers, suppliers and commerce service providers often must reinvent integration methodologies for their various trading partners," said Larry Mueller, president and chief operating officer for Ariba. "In order for B2B to scale to universal adoption and the now-famous trillion-dollar projections given by analysts, it's time for the industry to build on its early successes and collaborate on interoperability."

SAP Labs, Inc.:

"SAP is proud to be a core member of the UDDI initiative," said Heinz Roggenkemper, executive vice president of SAP Labs. "This initiative is in synch with the vision of mySAP.com. A standards based business and service directory is a fundamental building block for e-business collaboration. In addition, we are excited about the endless possibilities for our customers in greatly enhancing the visibility of their Internet services, as well as increasing overall business opportunities."

Sun Microsystems:

"Sun has always worked to help establish and support open, standards-based technologies that facilitate the growth of network-based applications, and we see UDDI as an important project to establish a registry framework for business-to-business e-commerce," said George Paolini, vice president of Java A Community Development at Sun. "As one of the primary contributors to the development of XML, Sun supports the planned use of this emerging standard as the data

foundation for the registry. We also applaud the UDDI project's stated intent to have the registry support services integrate with a wide set of existing core Internet technologies, such as Java, Jini ä, CORBA, RMI and HTML. Sun is looking forward to participating with all the UDDI member companies to create a truly open technology that helps companies take advantage of the power of the Internet for their business-to-business commerce."

5. Quellenverzeichnis

Bemerkung: zu UDDI gibt es noch keine gedruckte Literatur

Online Dokumente:

[1] UDDI Technical White paper:
http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.PDF

[2] UDDI Data Structure Reference V1.0
http://www.uddi.org/pubs/DataStructure-V1.00-Open-20000930_2.pdf

[3] UDDI Overview Presentation (PowerPoint Folien)
http://www.uddi.org/pubs/UDDI_Overview_Presentation.ppt

[4] UDDI Programmer´s API 1.0
http://www.uddi.org/pubs/ProgrammersAPI-V1.01-Open-20010327_2.pdf

[5] UDDI XML Schema
http://www.uddi.org/schema/uddi_1.xsd

Internetadressen:

[8] UDDI Hauptseite
http://www.uddi.org

[9] Linkseite zu UDDI
http://www.uddicentral.com/

[10] IBM UDDI Seite
http://www-3.ibm.com/services/uddi/

[11] Microsoft UDDI Seite
http://uddi.microsoft.com/

[12] IBM UDDI Browser
http://www-3.ibm.com/services/uddi/find

[13] SOAPClient UDDI Browser
http://www.soapclient.com/uddisearch.html

[14] Presse Artikel: ´The Book´ on UDDI
http://www.line56.com/articles/default.asp?NewsID=2531

[15] Presse Artikel: UDDI: A New Proposed Standard...
http://www.microsoft.com/presspass/features/2000/sept00/09-06uddi.asp