

Chapter 4: Universal Description, Discovery, and Integration

A search is a search, even if it happens to disclose nothing but the bottom of a turntable.

—Antonin Scalia

Overview

UDDI is in effect the publicity arm for Web services. It satisfies the self-advertising mandate of Web services. It is a standardized process to publish and discover information about Web services (as well as other services) programmatically or via a graphical user interface, which would typically be Web based. OASIS, which took over the stewardship for this technology as of July 2002, has a subheading in the “UDDI.org” section of its Web site that states: “Universal Description, Discovery, and Integration of Web Services.” Appending the two crucial words “Web services” to UDDI makes the goal of UDDI that much easier to perceive and appreciate.

A banner on that same OASIS Web page succinctly and elegantly highlights what UDDI is all about by saying: “UDDI is a key building block enabling enterprises to quickly and dynamically discover and invoke Web Services both internally and externally.” UDDI as such is a universal, totally open, platform-independent mechanism for describing, discovering, and integrating (i.e., consuming) Web services, though it can also be used to describe and advertise non-Web services—related services (e.g., the 1-800 toll-free, mail-order telephone number for a traditional brick-and-mortar business).

The mission of UDDI is to provide a standard, uniform service, readily accessible by applications via a programmatic interface or by people via a GUI, for describing and locating the following:

- Business and organizations that offer various services—Web services being one such service
- Meaningful description of the services being made available by the businesses and organizations listed as service providers
- Technical information as to how to locate, access, and utilize a particular service

As with all things related to or inspired by Web services, UDDI is highly XML-centric. The core information model used by UDDI, irrespective of the kind of service being described, is based on an XML schema. This UDDI XML schema deals with the following four types of key information as it relates to service providers and the services they offer, whether Web services or otherwise:

1. Business information pertaining to a business or organization providing one or more services—which, in the context of the UDDI model, is referred to as the “businessEntity”
2. Sets of related services (e.g., a set of Web services concerning purchase order processing and another set concerning online inventory checking) being offered by a previously described business or organization—which are referred to as “businessService” elements

3. The binding information necessary to invoke and make use of a particular, previously described, service (whether a Web service or otherwise)—referred to as a “bindingTemplate” element
4. More technical information (or even a technical blueprint) about a service, over and above the binding information necessary to connect to it, such as pointers to detailed specifications, protocols used by the service (e.g., SOAP, HTTP, or SMTP in the case of a Web service), and possible type classification for that service— which is known as a “technical model” (tModel)

There is a prescribed hierarchy among these four data structures. This hierarchy is shown in [Figure 4.1](#). These four core data structures have been a part of UDDI from the start. They form the nucleus of what UDDI is all about. Two other data structures have been added since, one in UDDI Version 2 and the other in Version 3. These two data structures deal with:

1. Relationships between business or organization entities (e.g., certification, alliances, membership, trading partnerships, and so forth), asserted to by one or both of those entities. This data structure, referred to as “publisher Assertion,” was introduced with UDDI Version 2.
2. Standing orders subscribed to by companies, organizations, or individuals requesting automatic notification in the event of a change to specific entities within the UDDI registry. This data structure, used for such automated change notification, which was introduced with UDDI Version 3, is referred to as “operationalInfo.” It is used, at present, to provide a change-order subscription mechanism. Consequently, this data structure is also sometimes referred to as “subscription.”

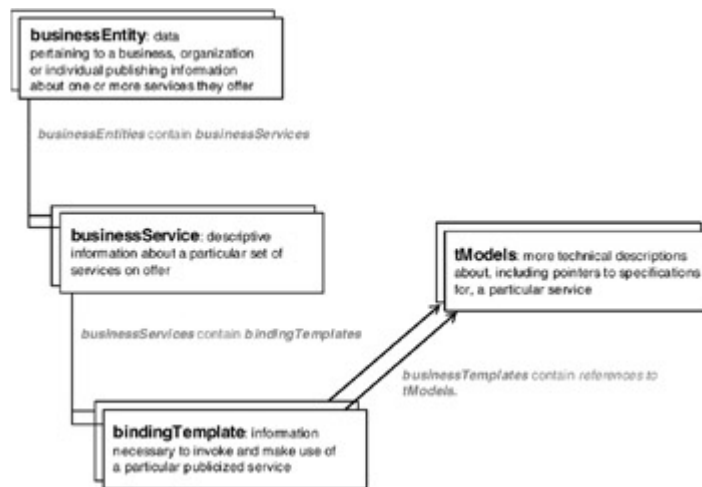


Figure 4.1: The four key UDDI data structures, based on the model shown in various UDDI documents.

[Figure 4.2](#) extends the data structure hierarchy shown in [Figure 4.1](#) to show how these two additional structures come into play.

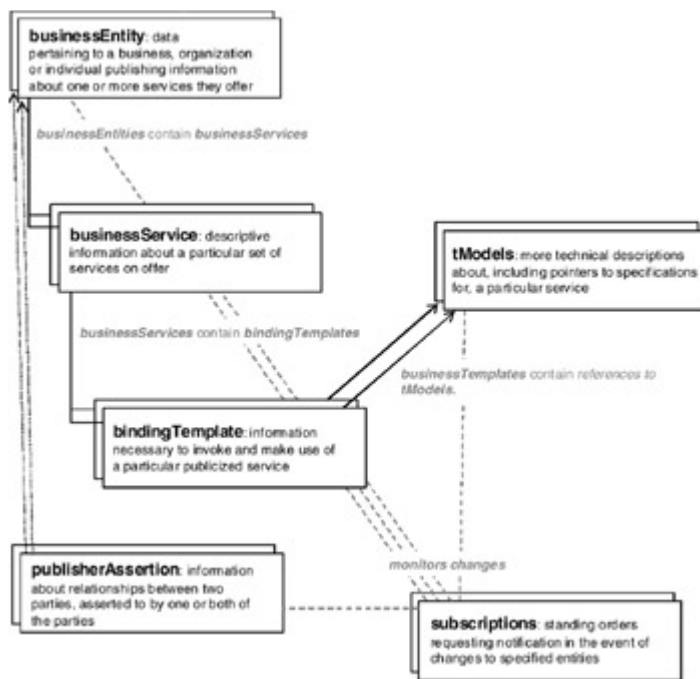


Figure 4.2: The four core and original UDDI data structures shown in Figure 4.1 extended to show the two additional structures that were introduced in the later versions of the UDDI specification.

The UDDI specification per se describes the XML schema for this information model, SOAP messages to transport the XML-based information, and a set of APIs to manipulate and manage the UDDI data. There are two key APIs: the UDDI inquiry API, which can be used to search or browse through the information contained in a UDDI Registry, and the UDDI publication API, which enables programmers to create or delete the information structures within a UDDI Registry. Invoking a UDDI API results in one or more SOAP messages being generated. There are about 40 inquiry and publishing functions—related SOAP messages that can be generated against a UDDI-compliant registry by the UDDI inquiry and publishing APIs.

UDDI, however, is not just a sterile specification. The UDDI instigators—namely, Ariba, IBM, and Microsoft—made sure that there would also be publicly accessible sets of implementations of the UDDI specification. They also made sure that other companies would actively buy into the UDDI initiative. They were successful in that endeavor, too.

In September 2000, when UDDI was initially unveiled, it was endorsed by about 30 blue-chip companies, including Sun, Compaq, Dell, American Express, Merrill Lynch, and Nortel Networks. Today this number is in excess of 200. UDDI is also a standards-based interoperable service available for free on the Web—with both a user-friendly graphical interface and a programmatic interface that conform to the UDDI APIs.

The two pivotal standards exploited by UDDI are XML and SOAP. Note that WSDL is not one of the prerequisite standards used by UDDI. In reality there is no formal relationship between UDDI and WSDL. They do, however, complement each other. Given that WSDL can be used to describe the interface of a Web service, there is obviously a role that WSDL can play vis-à-vis UDDI. The tModel for a Web service could thus point to a WSDL description, as could the bindingTemplate.

UDDI enables enterprises, individuals, and software applications to quickly, easily, incisively, and dynamically locate and obtain Web services, as well as other services. Though originally intended as a global public service available via the Web, the UDDI specification does not preclude private and

semiprivate implementations. The global public UDDI service, referred to as the Universal Business Registry or as the UDDI Business Registry (UBR), has been in operation since late 2000. The information maintained in the UBR can be accessed programmatically using a UDDI API, or via the Web-based user interface. [Figure 4.3](#) shows SAP's Web-based user interface to the UBR. The equivalent Microsoft and IBM interfaces to the UBR are shown in [Figures 1.7](#) and [1.8](#).



Figure 4.3: The Web-based user interface to SAP's UBR node.

The UBR was originally run by the three companies that collaborated to develop the initial UDDI specification, which was published in September 2000 as Version 1.0. Today, the UBR is being run by IBM, Microsoft, NTT Communications, and SAP—with Ariba, sadly, no longer involved. H-P, which was supposed to take Ariba's place when Ariba opted out, is also not a UBR player right now—partly due to the distraction of its merger with Compaq and partly to preclude the concept that the UBR was a wholly U.S. operation. In addition to the public Web-centric UBR, there is now also the concept of intranet UDDI implementations that serve just one enterprise, as well as extranet UDDI implementations that serve a group of enterprises collaborating with each other as partners.

Microsoft's new Windows Server 2003 includes a full-blown enterprise UDDI capability for realizing intranet or extranet implementations. IBM also offers a similar capability in its new WebSphere Application Server Network Deployment (WAS ND) V5—which has subsumed the previously available WebSphere UDDI Registry offering. In addition, UDDI-related development tools, some of which include private UDDI registries, are available from a wide variety of other vendors, including BEA, Sun, Novell, Fujitsu, IONA, and Cape Clear Software.

With the ready availability of these products and tools, there will now be a surge in intranet and extranet UDDI implementations as businesses and organizations gingerly start to evaluate the potential and power of Web services by using them in-house and in conjunction with trusted partners for pilot projects. Enterprise UDDI registries will enable these “not-ready-for-prime-time-as-yet” Web services to still be properly categorized and represented, using standard UDDI mechanisms, within the privacy of intranets and extranets. Recognizing the appeal and applicability of enterprise UDDI registries, the latest UDDI specification, namely, Version 3, sets out to specify the concept of UDDI registry interaction (i.e., how private UDDI registries can coexist and interoperate with the public UBR).

UDDI, given what it sets out to do, is a directory. It is a directory of businesses and organizations that provide various services. Thus, it is not surprising that UDDI is invariably thought of as a kind of electronic telephone directory—particularly an electronic yellow pages of sorts. In reality, UDDI is more, much more, than just a yellow pages directory that is organized purely by service type. UDDI also is a white pages directory in that it lists service providers by name. Consequently, the information

contained in UDDI is best thought of as being divided into three distinct categories, referred per telephone directory parlance as the following:

1. White pages
2. Yellow pages
3. Green pages

The white pages contain descriptive information about businesses and organizations providing various services. The business or organization name and a textual description of what they are will be included, where appropriate, in multiple languages. Contact information for that entity will also be included in terms of contact names, phone numbers, fax numbers, e-mails, and URLs. It will also list any known “industrial” identifiers, such as a Dun & Bradstreet (D&B) D-U-N-S nine-digit identification sequence, which uniquely identifies a particular business. The Thomas Registry identification scheme is another option.

The yellow pages, on the other hand, categorize businesses and organizations per industry-standard taxonomies. At a minimum, businesses and organizations will be categorized in terms of their industry, the products and services they offer, and their geographic location. The North American Industry Classification System (NAICS), which has superseded the U.S. Standard Industrial Classification (SIC) system, is one of the taxonomies that can be used to specify industry classification. Similarly, the United Nations Standard Products and Services Code System (UNSPSC) can be used to specify product/service classifications. UDDI is also innately extensible. Thus, it permits the use of new taxonomies provided that all users of the UDDI Registry have a means of interpreting the classification scheme used.

The green pages are where the technical information needed in order to use an available service is catalogued. The green pages, in effect, contain the information contained in the bindingTemplate and tModel data structures for a given service—as listed in a businessService entry (as shown in Figures [4.1](#) and [4.2](#)). To wrap up this phone book analogy, note that the white pages contain the information found in the businessEntity data structures of a UDDI Registry, whereas the yellow pages categorize, per accepted taxonomies, the information maintained in the businessService data structures. The green pages, then, augment the entries in the yellow pages by including the information found in the bindingTemplate and tModel structures for that particular service. [Figure 4.4](#) depicts the UDDI information content in terms of the phone book model, whereas [Figure 4.5](#) correlates the phone book model to the UDDI information hierarchy shown in [Figure 4.1](#).

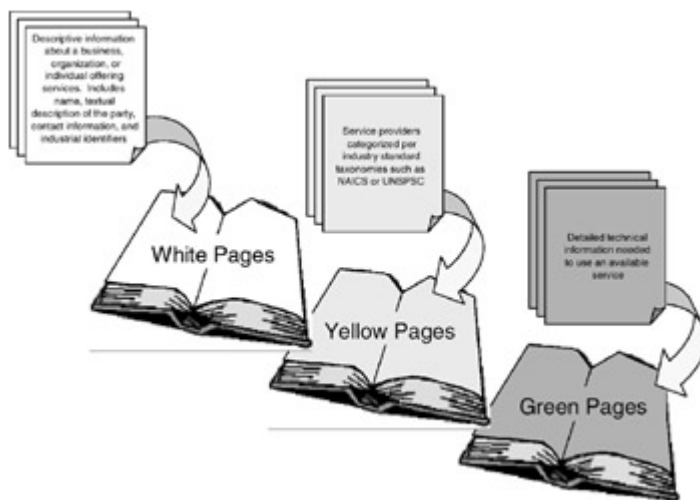


Figure 4.4: The UDDI information content per the telephone directory model.

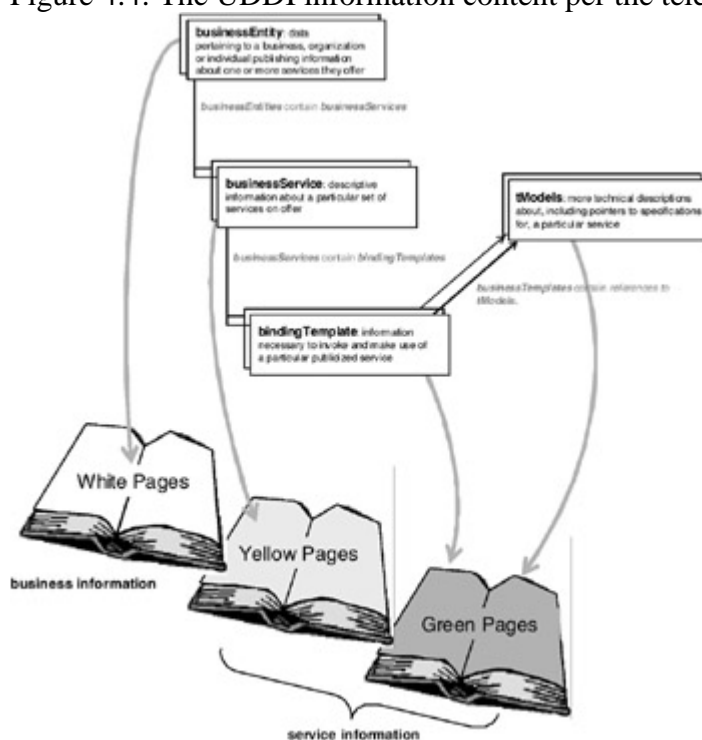


Figure 4.5: Correlating the UDDI telephone directory model with UDDI's four core data structures.

The bottom line here, for the time being, is that UDDI in general, and the public UBR in particular, are cornerstones of the overall Web services initiative. Given that it provides the search mechanism for locating applicable Web services, the eventual success and popularity of Web services are obviously going to be contingent, to a large degree, on the efficacy of UDDI. Given that the deployment and adoption of Web services have yet to live up to expectations, it is not surprising that the UBR, at present, is not brimming with germane entries.

Nonetheless the volume, the categories, and the amount of information per entry has been slowly but palpably growing since the summer of 2002. The availability of full UDDI functionality, as a standard, no-charge feature within Microsoft Windows Server 2003, will further increase UDDI awareness and promote enterprise-level implementations. Based on these trends, one has to assume that by 2010, professionals in the business community will often stop to ponder how they used to locate businesses and services prior to the advent of UDDI—in much the same way that percipient people today often

wonder: How did we manage to do *that* before the Internet?



4.1 The rationale and motivation for UDDI

UDDI was conceived at the time that the dot.com frenzy was building to its crescendo. E-business seemed unstoppable as the long-sought-after panacea for expediting and streamlining commerce. Against this backdrop, UDDI was initially envisaged as a standards-based, vendor-neutral mechanism for globalizing and simplifying Web-based, electronic b2b interactions. UDDI would automate many of the discovery and integration aspects of an eventual e-business transaction—particularly those having to do with locating the appropriate partners (e.g., suppliers in the case of an SCM scenario) and the mechanisms by which these potential partners conduct business.

UDDI would eliminate the guesswork and chance that was then an inherent part of e-business, given the lack of globally accepted directories for listing e-business participants. The main issues that UDDI intended to address included the following:

- How an enterprise, whether a business or organization, identifies other enterprises around the world that may be able to provide it with necessary services (or goods)
- How enterprises could describe themselves and the services they offered in a structured, systematic manner so as to attract prospects interested in such services
- How an enterprise could sift through and narrow down its list of prospective service providers to those that best fit its needs, based on specific criteria, given that unbounded searches on the Web could produce very large and unwieldy lists of results, considering the millions of enterprises that maintain a high-profile presence on the Web
- How one could obtain an accurate, detailed description of the services being offered by a selected enterprise
- How one could determine the mechanisms available for conducting e-business transactions with a selected enterprise
- How one could realize all of the above using programmatic interfaces—rather than having to do all of this manually using a user interface

UDDI strived to be as generic, inclusive, and flexible as possible when it came to the services it would accommodate. Though it wanted to promote Web services, it went to great lengths to ensure that its scope would not be restricted just to Web services. It wanted to facilitate all possible means of b2b, with Web services being just one, albeit a very strategic one, of the available mechanisms. The UDDI-based search and discovery process can thus be gainfully used, in theory, to realize the means for achieving any and all types of e-business transactions—hence, the reason why UDDI and the UBR are not limited purely to Web services.

Consequently, there is nothing to preclude a business that offers no Web services whatsoever from still publicizing the services it does offer (e.g., mail order service, consulting services, shipping services,

import/export expertise, or financial services) on the UBR. This unalienable fact that UDDI is not restricted to just Web services is something that one should always keep in the back of one's mind when dealing with Web services— or UDDI-related issues.

The obvious question at this juncture is why the UDDI instigators, in particular IBM and Microsoft, did not pursue the option of extending the then-rampant Internet search engines (e.g., Yahoo!, Alta Vista) to include the envisaged UDDI functionality. Another option would have been to try to enhance one of the then-nascent e-business service registries, such as the ones being promulgated by WebMethods or CommerceOne, to fulfill the UDDI goals. The cynical and first blush answer to this would be that both these superpowers in the IT world, much to their chagrin, had “no skin” in search engine technology or in e-business registries. However, to be fair, search engine technology, then as it still is now, was unregulated, proprietary, and not based on any accepted standards other than HTML and URLs. The e-business registries were also proprietary and too closely associated with specific vendors.

The UDDI instigators wanted UDDI to be as standards based as Web services. Standards compliancy and, through that, vendor neutrality and platform independence were the main planks of the Web services platform. To sustain this value proposition, UDDI, as an enabling technology for Web services, also needed to be entirely standards compliant. In addition to the absence of relevant standards that governed their operation, some of the other key factors that made search engines unsuitable for the purposes of UDDI included:

- Search engines devoted all their efforts to locating and indexing just the URLs of the Web pages that contained keywords of interest. Search engines did not make any effort to identify other identification or contact information such as e-mail addresses, FTP sites, or even phone numbers. Search engines, even today, are very HTML-centric and URL specific. UDDI was seeking a more generalized search-and-discover mechanism.
- Search engines specialize in doing free text searches against unstructured data, whereas UDDI wished to promulgate structured data models.
- Search engines, in general, do not offer a no-charge, self-service, publishing mechanism whereby enterprises or individuals can enter specific information that they wish to publicize.
- Search engines, even today, do not typically offer programmatic access to the services they offer.
- Search engines have yet to become XML savvy.

Given all the previous limitations of search engines, it is understandable that the UDDI instigators wanted to create something that was considerably more compatible and consistent with the methodologies that they were in the process of introducing for Web services—in particular, SOAP and XML. At this juncture SOAP had just been introduced, so there were no SOAP-related standards per se that might have been of relevance for realizing UDDI. Therefore, there really was no other viable option than to create UDDI from scratch around SOAP and XML schema—and then put it forward as a proposed standard. That is exactly what happened.

At this point, those who are familiar with other directory schemes may be wondering, quite legitimately, why the OSI directory standard X.500 or the Lightweight Directory Access Protocol (LDAP), its equivalent in the TCP/IP world, could not have been used to achieve the goals of UDDI. The reality here, however, is that UDDI is meant to be at a higher level, a more logical level, than these two directory setup and directory access-oriented standards.

The parallel here is that of SOAP being developed as a transport-agnostic messaging scheme that can be used on top of HTTP, RPC, TCP, SMTP, FTP, message queuing, and so forth. Similarly, UDDI set out to define XML schema-based data models and APIs that generate SOAP messages, which, in theory, could be implemented on top of a standard directory scheme. Thus, it is indeed possible to build a fully conformant UDDI Registry on top of LDAP or X.500—and there are indeed specifications on the Web as to how such implementations can be realized. BEA even offers an enterprise implementation solution built on top of LDAP. That said, it is common knowledge that the UBR nodes, at present, are implemented using popular relational database systems (e.g., DB2 in the case of IBM).

Given this background, the advantages that UDDI bestows upon the global business community at large can be summarized as follows:

1. Provide businesses, organizations, and even individuals with an unprecedented, no-charge, totally egalitarian mechanism to showcase all of their services in a systematic way to the entire world—thus enhancing their market reach, competitiveness, marketing efficacy, and overall business prospects
2. Provide a very structured and consistent mechanism for locating and obtaining necessary details about pertinent services available on the global market
3. Facilitate businesses in identifying prospective b2b partners
4. Enable service providers and services to be registered, as well as located and invoked, using standardized, programmatic interfaces, thus bolstering the electronic infrastructure for the next generation of e-business
5. Simplify and streamline e-business-related software integration by ensuring that access to all of the pertinent technical information is available over the Web via a standard mechanism



4.2 The UDDI model

The “UDDI Version 3 Feature List” document, which was published by UDDI.org in mid-2002, starts off with this pithy (but uncharacteristically Web services–centric) description: “UDDI Version 3 builds on the vision of UDDI: a ‘meta-service’ for locating Web services by enabling robust queries against rich meta-data.” Note that the term *meta* gets used not once but twice in this somewhat short description, with the first mention, intentionally or otherwise, having two very appropriate and germane connotations in the context of UDDI.

Technical “Webizens,” whose lives revolve around the Web, will immediately interpret “meta-data,” in this context, as referring to it being machine readable—in other words, data that are programmatically accessible, or in this case data that are programmatically searchable. This is consistent with the W3C’s definition of meta-data, which happens to be: “Metadata is machine-understandable information for the Web.”

It also ties in with HTML’s <META> tag, which enables one (among other things) to specify Web site identification and keyword information to help search engine Web crawlers to more easily categorize

and index that Web page. Again, the concept here is that of automatic, programmatic searches. This programmatic aspect, as indicated by this meta-data reference, as should now be clear, is a fundamental plank of UDDI.

The prefix “meta,” however, has a more traditional, pre-Web connotation. It is used to denote a description that is one level higher than the word being prefixed. Thus, if “x” is some entity, then meta-x is a description of “x.” Therefore, a metasyntax is a syntax for describing other syntaxes. Similarly, a meta-language is a language that can be used to describe the makeup of other languages.

Hence, this is why XML is referred to as a meta-markup language. It allows one to describe the composition of other markup languages or, more commonly, to describe data that would otherwise be annotated with another, lower-level markup language. This now brings us to UDDI being a “meta-service.” Given that UDDI is modeled around APIs that generate SOAP messages, UDDI definitely falls into the category of machine-understandable service—if one were to extrapolate the W3C meta-data definition to cover meta-service.

However, UDDI, being a meta-service, also alludes to it being a service for locating other services—particularly Web services. Thus, this new mission statement, included with a UDDI Version 3 must-read overview document, hits the nail on the head quite squarely by highlighting the “service about other services” and programmatic aspects of UDDI. Relative to this backdrop, the initial UDDI model postulated with UDDI V1, in September 2000, is shown in [Figure 4.6](#), whereas [Figure 4.7](#) shows the multiregistry structure that is likely to be common in the future. This basic model still holds true for the UBR. The one big difference that has taken place since, particularly with UDDI V3, is the concept of the UBR being augmented by private and semiprivate UDDI registries. However, the non-UBRs also still conform to the overall precepts of the original model, albeit with less external entities participating in populating, categorizing, and interrogating a given private or semiprivate registry.

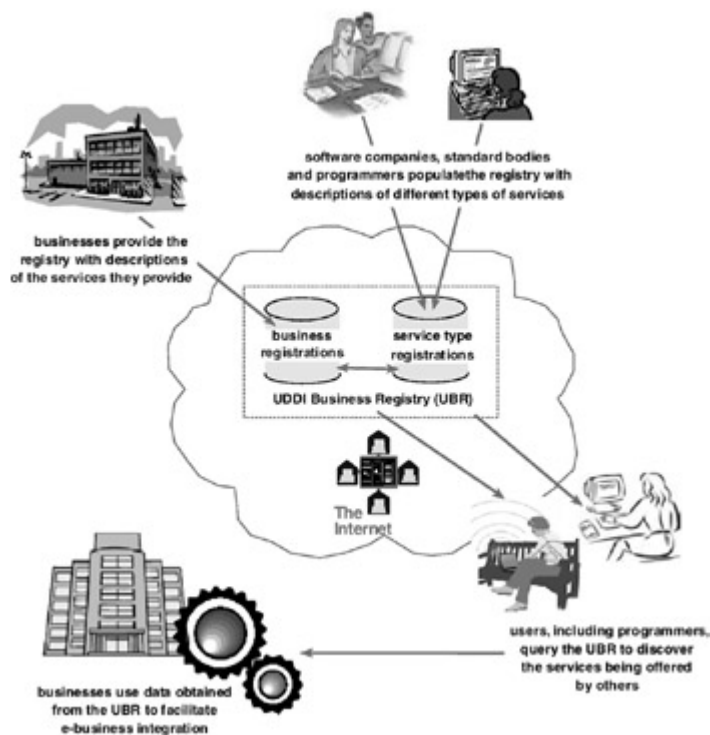


Figure 4.6: A simplified and rationalized version of the original “How UDDI V1 Works” model postulated by UDDI.org in September 2000.

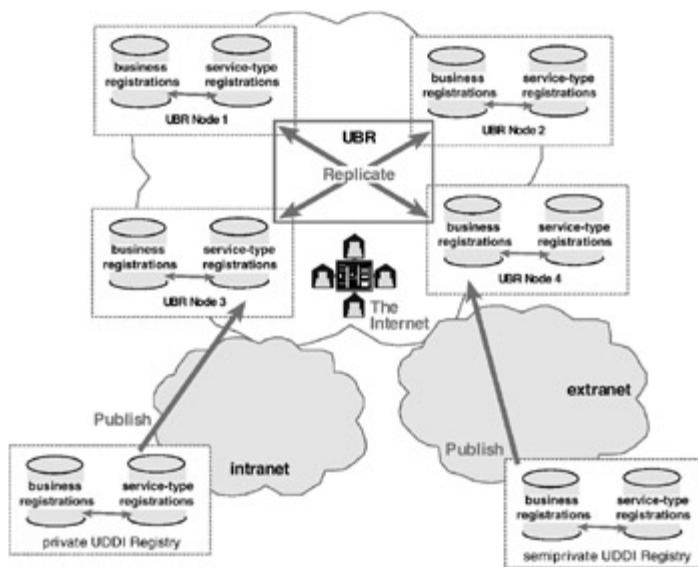


Figure 4.7: Post-2003 UDDI scenarios, where the UBR will be complemented by affiliated private and semiprivate UDDI Registries.

4.2.1 UDDI data structures

The UDDI information model is composed of instances of persistent data structures. A persistent data structure is one that automatically preserves its old versions. With a persistent data structure, one can make changes to the structure without destroying the old version. Thus, all versions of the structure persist. This enables previous versions of the structure to be queried, in addition to the latest version. The use of persistent data structures by UDDI means that one could have access to different versions of a service rather than just to the most recent.

UDDI's persistent data structures are referred to as “entities.” An “entity” in XML is a special unit of storage. Entities are the basic building blocks of an XML document. Though UDDI does not make explicit reference to this connection, it is implied and will be appreciated by those familiar with XML. The UDDI information model is said to be composed of instances of six core entity types. These entity types, which are shown in [Figure 4.2](#), are as follows:

1. businessEntity
2. businessService entity
3. bindingTemplate entity
4. tModel entity
5. publisherAssertion entity
6. subscription entity

All these entities are represented in XML, with each structure corresponding to predefined XML elements and XML attributes. They are stored, in a persistent manner, in one or more UDDI nodes. Each entity acquires the type of its outermost XML element.

There is a very definite and immutable hierarchy between these six UDDI data structures, as depicted in [Figure 4.2](#), though if one wants to be ultra-pedantic, one would claim that this hierarchy relates to just the four original UDDI V1 structures, with publisherAssertion and subscription being more in the nature of “flags” or references that apply to the other structures. Each business or organization described in a UDDI Registry exists as a separate instance of a businessEntity data structure. Similarly, each service offered by a business or organization is maintained as a separate instance of data.

The fixed hierarchy between the UDDI data structures also results in an automatic, corresponding containment relationship between these structures. Thus, the businessEntity structure contains one or more unique businessService structures, whereas each businessService structure, in turn, contains specific instances of bindingTemplate data. The bindingTemplate structures will contain information that includes pointers (or references) to specific instances of tModel structures.

Within the UDDI hierarchy, no single instance of one of the core structure types is ever contained by more than one parent structure. Thus, as clearly depicted in [Figure 4.2](#), only one specific businessEntity structure will ever contain information about a specific instance of a businessService structure. At this juncture it is best to look at a specific data structure in detail so as to reinforce what has been discussed so far—as well as to provide a basis for discussing a set of new UDDI concepts and constructs. [Figure 4.8](#) shows the detail composition of the businessEntity structure. [Figure 4.9](#) shows how an instance of data for a particular service provider (in this case IBM), as maintained per this structure, would be presented to a user of a UBR node.

```
<element name="businessEntity" type="uddi:businessEntity" />
<complexType name="businessEntity">
  <sequence>
    <element ref="uddi:discoveryURLs" minOccurs="0" />
    <element ref="uddi:name" maxOccurs="unbounded" />
    <element ref="uddi:description" minOccurs="0" maxOccurs="unbounded" />
    <element ref="uddi:contacts" minOccurs="0" />
    <element ref="uddi:businessServices" minOccurs="0" />
    <element ref="uddi:identifierBag" minOccurs="0" />
    <element ref="uddi:categoryBag" minOccurs="0" />
  </sequence>
  <attribute name="businessKey" type="uddi:businessKey" use="required" />
  <attribute name="operator" type="string" use="optional" />
  <attribute name="authorizedName" type="string" use="optional" />
</complexType>
```

Figure 4.8: The businessEntity structure, as defined in the “UDDI V2.03 Data Structures Specification,” in relatively easy-to-follow XML schema notation, showing which elements or attributes are required (i.e., use=“required”), those that can be repeated (i.e., maxOccurs=“unbounded”), and those that are optional (i.e., minOccurs=“0”).



Figure 4.9: The business provider information for IBM, as it appears in Microsoft’s UBR node, illustrating how data maintained in a businessEntity structure, as shown in Figure 4.8, get utilized. The long, hexadecimal Provider Key, which appears near the middle of the Web page, corresponds to the unique business key identifier for this entry. Also note the tabs for services, contacts, identifiers, categories, and discovery URLs, which correspond directly to elements within the businessEntity structure. The Name entry, which appears after the first blocked-out horizontal bar, in this instance, only has one entry—and that is in English, as denoted by the (en). It would have been possible to have multiple entries here.

Most of the elements in the businessEntity structure, such as “name,” “contacts,” “discoveryURLs,” “businessServices,” and so forth, are self-explanatory. Some, in particular “identifierBag” and “categoryBag,” can, however, be somewhat obscure or misleading. The postfix “Bag,” in this case, was probably not the best word to use in this context. The term “bag” in this context strives to convey that it is possible to have a mixed list (i.e., a mixed bag) of identifiers or categories—albeit with both of these lists being optional. What is listed in the identifierBag are accepted industrial identification “numbers” as D-U-N-S, nine-digit identification codes. [Figure 4.10](#) shows the D-U-N-S identifier for IBM as it appears in Microsoft’s UBR node.



Figure 4.10: The D-U-N-S, nine-digit identification for IBM shown in the Identifiers tab as it appears within IBM’s entry at Microsoft’s UBR node.

Business or organization categories as they relate to a yellow-page taxonomy are listed in the categoryBag. [Figure 4.11](#) shows the start of the list of 170 categorizations that IBM lists for itself using a combination of NAICS and UNSPSC codes. IBM’s comprehensive list of categorizations of what it does ranges from “optical jukebox server software” to “sales financing.” It also includes the UNSPSC codes for “spreadsheet software,” “word processing software,” and “presentation software.”



Figure 4.11: The start of the 170 categorizations IBM lists for the services and products it offers using a combination of NAICS and UNSPSC codes. On the other hand, Microsoft, uncharacteristically self-effacing, only lists one category—that being the NAICS “51121” code for a “software publisher.”

All of these categorizations are germane, and one has to applaud IBM for its diligence in coming up with all of these codes. However, if you look at Microsoft’s listing of categories, you will be surprised just to find one! That is the NAICS code for a software publisher. IBM happens to list that one, too. This highlights one of the unintended, but inevitable, shortfalls of UDDI. UDDI is unregulated, self-policing, and essentially unaudited.

The completeness and accuracy of the entries are left entirely to the discretion, professionalism, and motivation of the business or organization entering the data about itself. Until Microsoft gets around to updating its list of categorizations, there will be the irony that somebody doing a UDDI query for providers of word processing software will find IBM as a search result, but not that of the undisputed market leader Microsoft!

4.2.2 UDDI keys

Each entity in a UDDI Registry, whether a business, a service, a service binding information, or a tModel, is assigned a specific UDDI identifier or UDDI key. An example of a UDDI key, which is actually my consulting services–related entry in the UBR, would be:

```
uddi:E05BE6A0-C043-11D7-9B41-000629DC0A53
```

[Figure 4.12](#) is a screen shot from IBM’s UBR node showing the businessEntity entry for “Anura Guruge,” with the above key prominently displayed as the first item of information—albeit not showing the mandatory “[UDDI](#)” key. Compare this businessEntity presentation à la IBM with that shown in [Figure 4.9](#), which depicts how such information is presented by Microsoft. This is something that you have to get used to when dealing with UDDI implementations—including the global UBR.



Figure 4.12: A businessEntity presentation for another service provider, in this case for Anura Guruge, as displayed on IBM’s UBR node, showing the 32-digit hexadecimal key for this entry. Compare this presentation with how Microsoft displays businessEntity information, shown in Figure 4.9, and note that although IBM just refers to the UDDI key as the “key,” Microsoft calls it the “Provider Key” and uses lowercase to display the hexadecimal code, whereas IBM opts for the more traditional uppercase.

The UDDI specification only deals with the information models, the APIs, the XML schema, and the SOAP messages. It does not talk about implementation details, so each UDDI implementation, including the various nodes that make up the global UBR, can have a different look and feel. However, all nodes of the UBR, given that they are working with the same set of information, uniquely identified by the UDDI keys, will always present the same information content—as well as information instances.

The UDDI key uniquely identifies that entity, for the duration of its existence, within the UDDI Registry. The persistence of UDDI data is related to the use of these unique keys. In the case of the UBR, all the nodes that make up the UBR will use and reflect the same UDDI key for a specific instance of information, given that all the nodes that make up the UBR work in concert when it comes to the information maintained by the UBR.

The “Provider Key” in [Figure 4.9](#) is also an example of a UDDI key—in this case the key to this particular instance of IBM’s businessEntity information. [Figure 4.13](#) shows IBM’s entry, as displayed on IBM’s UBR node as opposed to Microsoft, to confirm that the provider key is still the same irrespective of the UBR node being used—though it looks different because Microsoft displays the hexadecimal digits in lowercase per the wont of XML and the Internet, whereas IBM, the godfather of hexadecimal notation since the early 1960s, opts for the more common uppercase-only representation. This is another example of how implementation details will vary from node to node, thus giving some users pause for thought if they skip from one implementation to another. UDDI keys, as with all XML-related entities, are, nonetheless, case sensitive, though this case sensitiveness does not apply to whether hexadecimal digits appear in lowercase or uppercase, given that these are in reality numeric digits, though shown using letters.

The screenshot displays the IBM UDDI Business Registry interface. The main heading is "UDDI Business Registry Version 2" with the subtitle "Universal Description, Discovery and Integration". The page is titled "Business Details" and includes a note: "The details of the selected business are shown below. Please use your browser's Back button to return to the previous page OR Press the New Search button to search again."

Business Information

Key	Usage
0093110-3A4F-1125-80C0-06205220C68	BusinessEntity

Discovery URL: <http://uddi.ibm.com/uddi/v2/BusinessEntity-0093110-3A4F-1125-80C0-06205220C68>

Business Name(s)

Name	Language
IBM Corporation	en

Business Description(s)

Description	Language
All IBM, we strive to lead in the creation, development and manufacture of the industry's most advanced information technologies, including computer systems, software, networking systems, storage devices and microelectronics.	en

Business Contact(s)

Name	Role
IBM Corporation	general
IBM Corporation	corporate offices
IBM Corporation	US general
IBM Corporation	supplier

Business Location(s)

Code	Description	Type
43.18.25.68.00	UNSPSC: Storage media loading software	UNSPSC v7.2
43.18.25.68.00	UNSPSC: General utility software	UNSPSC v7.2
43.18.26.01.00	UNSPSC: Platform interconnectivity software	UNSPSC v7.2
43.18.26.02.00	UNSPSC: Optical jukebox server software	UNSPSC v7.2
334	NAICS: Computer and Electronic Product Manufacturing	NAICS

Figure 4.13: BusinessEntity display for IBM, at IBM's UBR node, confirming that the UUID key for this entry is exactly the same as that shown in Microsoft's UBR node, per Figure 4.9, with IBM, however, displaying the hexadecimal digits in uppercase versus the lowercase depiction preferred by Microsoft. Also note that IBM's presentation includes the business description, as well as the categorizations (i.e., Business Locator[s]), and that these categorizations are listed in the same order shown in Figure 4.11.

A UDDI Registry assigns one of these unique identification keys when a new registry entry is first published (i.e., saved). Once assigned, a UDDI key can be used at any later time to incisively access that specific instance of data—on demand. Prior to UDDI V3, all UDDI keys had to be generated by the “publishing” UDDI node to ensure the uniqueness of each key. UDDI nodes generated these unique keys in the form of Universally Unique Identifiers (UUID). A UUID, also called a Globally Unique Identifier (GUID), is a scheme that permits resources to be uniquely named over time and space.

UUIDs were originally created for use with remote procedure call (RPC) mechanisms in the context of the Network Computing System (NCS) initiative of the 1980s. This RPC mechanism was later adopted by the Open Group's Distributed Computing Environment (DCE) and further formalized by ISO within the framework of the once-rampant Open Systems Interconnection (OSI) model as ISO standard: ISO/IEC 11578:1996.

A UUID is 128 bits long. It can thus be represented by 32 hexadecimal digits, given that each hexadecimal digit corresponds to 4 bits. The UUIDs used as UDDI keys are depicted as 32-digit hexadecimal sequences by the current UBR implementations, as illustrated in Figures 4.9, 4.12, and 4.13. A UUID is guaranteed to be different from all other UUIDs generated until 3400 A.D., or, worst case, still likely to be extremely different from any other even if the UUID generation algorithm is not optimally implemented.

The algorithm used to generate UUIDs relies on a combination of hardware addresses (derived from IEEE 802 MAC addresses uniquely assigned to each and every LAN adapter), timestamps, and random seeds. It is the use of timestamps that results in the caveat that the uniqueness of UUIDs is only guaranteed up until 3400 A.D.! The 32-hexadecimal digit UUIDs, the only type of keys available with UDDI V1 and V2, are not well suited for use by people. They are arbitrary, unwieldy, and impossible to remember. With UDDI V3, it is possible to have keys that are considerably better suited for use by people. The creator (i.e., publisher) of a UDDI entity can even request a particular key or a type of key. These so-called “publisher-assigned keys” are one of the major new features of UDDI V3.

With V3 it is now possible to have so-called “domainKeys” in addition to the prior “uuidKeys.” Domain keys are based on the now-commonplace Internet domain names. Since domain names have to be unique, keys based on domain names can also maintain UDDI’s requisite individualism. A domain name-based UDDI key would look like this:

```
uudi:acmecompany.com:businessRegistration
```

Thus, a domain name-based UDDI key for my businessEntity could be:

```
uddi:inet-guru:AnuraGurugeBusinessName
```

Domain name-based UDDI keys are obviously much more flexible and meaningful than uuidKeys. However, the UDDI specification leaves as an implementation option whether a given UDDI Registry has to support domainKeys. In addition to these two orthogonally different key types, it is now also possible to have what are referred to as “derived keys.” A derived key has either a uuidKey or a domainKey as its base (i.e., its start), but has in addition an alphanumeric ASCII character string, or arbitrary length, appended to it. This, especially in the case where a uuidKey is the base, provides further flexibility when it comes to UDDI key assignment.

The growing trend toward having enterprise UDDI registries, whether private or semiprivate, augmenting the UBR, does complicate the management of UDDI keys—especially if domainKeys are in use. The issue has to do with UDDI publishers that wish to copy the entire contents of a UDDI entity entry from one UDDI registry (e.g., private) to another while preserving the key assigned to the original entity. UDDI V3 includes a capability known as “entity promotion” to cater to such scenarios. With entity promotion, a UDDI publisher is permitted to propose the existing key for an entity as the preferred key for that entry. It is, however, left to the implementation and the exact policies of a given registry as to whether the new registry has to acquiesce to this request.

To facilitate publisher-assigned keys, in particular entity promotion scenarios, V3 introduces the concept of root and affiliate UDDI registries. The use of a root UDDI registry by multiple affiliate registries ensures that the affiliate registries can readily share data among themselves, relative to that root, and still make sure that they maintain unique UDDI keys, since this would be arbitrated by the root. It is suggested that the UBR should be considered, whenever possible, as the root registry in such distributed registry scenarios. In this case the UBR, through the use of either uuidKeys or domainKeys, will ensure the uniqueness of the necessary UDDI keys.

4.2.3 UDDI APIs

The UDDI inquiry and publication APIs, as mentioned earlier, are key to any UDDI implementation. They control the core functions related to publishing and locating entities in a UDDI Registry. Though presented so far as single APIs for simplicity, the UDDI specification treats all APIs as consisting of sets. It then further splits these sets into node versus client API sets. The API sets specified in V3 are

thus as follows:

UDDI Node API Sets

1. UDDI Inquiry
2. UDDI Publication
3. UDDI Security
4. UDDI Custody Transfer
5. UDDI Subscription
6. UDDI Replication

UDDI Client API Sets

1. UDDI Subscription Listener
2. UDDI Value Set

The functionality made possible by these UDDI API sets can be characterized as follows:

- **UDDI Inquiry:** This API set is what a programmer would use to locate and obtain details on specific entries stored in a UDDI Registry. This API set supports three distinct patterns of inquiry to address all germane inquiry modes for this type of data. The three types of inquiry patterns supported are referred to as browse pattern inquiry, drill-down pattern inquiry, and invocation pattern inquiry. A browse pattern inquiry, as the name alludes, permits one to conduct an inquiry starting with a broad categorization and then refining the inquiry to more specific criteria as each set of search results is displayed. A drill-down inquiry involves using a UDDI key, obtained via a browse pattern inquiry or by other means (e.g., using a domain-Key), to access a specific entity. An invocation pattern inquiry is typically used in the context of bindingTemplate entries for Web services. As suggested by the name, it is used when the application invoking this search is happy to have the Web service located by the search to be dynamically invoked at the end of the search, so this becomes a search-and-activate operation.
- **UDDI Publication:** This API set is used to publish, update, and delete information contained in a UDDI Registry. It is left to the implementation policies of a particular registry to decide which node of a particular UDDI registry is used by a publisher to publish a given set of data. UDDI per se does provide an automated mechanism to check and reconcile information that may be published on different nodes of the same registry at different times. This again is an implementation issue.
- **UDDI Security:** This API set is currently used to obtain and manage authentication information.
- **UDDI Custody (and Ownership) Transfer:** The publisher who initially creates an entry has ownership of that entity. A custodial UDDI node (typically the one at which the entry was published) has to maintain a rigid relationship between an entity and its owner to ensure the integrity of the information in a UDDI Registry. In the case of a multinode registry (e.g., UBR), every node must guarantee the integrity of an entity's ownership and custody. Consequently, a node cannot permit changes to an entity unless it is the custodial node for that entity. The Custody (and Ownership) Transfer API set enables one node to transfer custody of certain entities to another node or to transfer ownership of an entity from the current owner to a new one—in an authorized and cooperative manner.
- **UDDI Subscription:** This is the API set that facilitates the subscription scheme introduced with UDDI V3. This API set enables clients, referred to in UDDI argot as subscribers, to register their interest in being notified if and when changes are made to specific entries in a registry.
- **UDDI Replication:** Used to replicate entries within a UDDI Registry.

- **UDDI Value Set:** This API set permits authorized third parties to validate certain information being published in a UDDI Registry based on registered “value sets.”

4.2.4 UDDI nodes and UDDI registries

A software system that supports at least one of the UDDI node API sets described earlier is considered to be a UDDI node. The UDDI V3 specification tries to put some “self-serving” spin into this definition by stating: “A set of Web services supporting at least one of the node API sets is referred to as a UDDI node.” What is noteworthy here is the introduction of the concept of Web services supporting the UDDI API sets. That is the self-serving part, since one can justifiably argue that it is not necessary to include Web services in this definition. On the other hand, some technocrats will argue that this is indeed a valid and necessary distinction, based on the fact that UDDI relies on SOAP.

The UDDI APIs, by definition, operate by generating SOAP interactions. Thus, any software system that supports a UDDI API set, by default, has to be able to accommodate SOAP. SOAP is considered by most, including the nascent W3C Web Services Architecture, to be a fundamental (if not a prerequisite) building block of Web services. When one subscribes to this SOAP-centric view of Web services, one could then go the next step and describe any SOAP-based piece of software as being a Web service.

At a minimum, one could claim that any piece of software that communicates using SOAP is Web services ready. At this juncture, all that is important to note is that the UDDI definition for a UDDI node calls for the node to support at least one of the node API sets. Whether the software that makes this possible is a set of Web services or not is not that critical, provided there is an appreciation of what is being alluded to here.

In addition to supporting at least one node API set, the three other criteria to which a UDDI node has to conform are as follows:

1. It interacts with UDDI data via the appropriate UDDI API sets.
2. A UDDI node can only be a member of exactly one UDDI Registry.
3. A UDDI node, at least conceptually, has full access to and can manipulate data structures of a complete logical copy of the total data managed by the registry of which they are a part. All interactions made via that node’s query or publish API sets must always apply to these data—and no other (i.e., the UDDI APIs should only be used to access and manipulate the data associated with the UDDI Registry).

A UDDI Registry is made up of one or more UDDI nodes. When a registry is comprised of multiple nodes, all the nodes have to work in a collaborative manner to ensure that all nodes have access to the same logical set of data. The nodes making up a registry are collectively responsible for managing the UDDI data associated with that registry. The nodes will typically achieve this goal by using UDDI replication between the nodes using the UDDI Replication API set.

As previously stressed, the UDDI specification does not specify any implementation criteria for realizing a UDDI node or a UDDI Registry. Thus, the UBR was implemented, from day one, as a multinode registry. However, most private (if not semiprivate) registries are likely, at least to begin with, to just consist of one node. How a given registry implements policy decisions (e.g., authentication, integrity, support of digital certificates, and so forth) is also left as an implementation option. However, the necessary policy decisions have to be implemented in a consistent manner at each and every

applicable point within the registry. A registry, however, has the option of delegating certain policy decisions to individual nodes—which is true in the case of the UBR.

[Figure 4.14](#) brings this architecture-related section to a close by showing, in a “stack”-oriented depiction, how and where UDDI fits in relative to the other Web services–related methodologies.

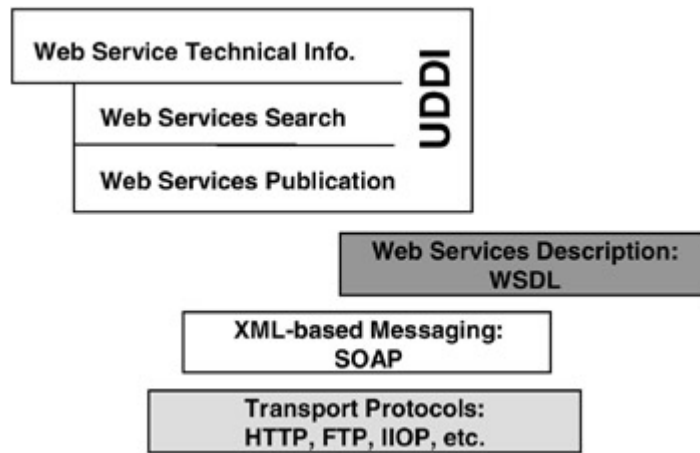


Figure 4.14: The Web services stack showing the basic relationship between the various Web services enabling technologies and UDDI.

Team LiB
Team LiB

◀ PREVIOUS NEXT ▶
◀ PREVIOUS NEXT ▶

4.3 How UDDI has evolved

Even by today’s Internet-fueled pace, the evolution of UDDI has been relatively rapid and pronounced. There were only 22 months between UDDI V1 and V3. The overriding motivation to make UDDI a bona fide industry standard as soon as justifiably possible explains some of the haste to push out new specifications way ahead of the actual technology adoption curve. There is nothing wrong in this, and the key UDDI instigators—namely, IBM, Microsoft, and Ariba—need to be applauded for their commitment and drive.

When the UDDI Consortium was created in the summer of 2000 to bring forth UDDI V1, its charter was to oversee UDDI through V3. At that juncture UDDI was to be handed over to an accepted standards body. This indeed was exactly what happened, with the Consortium handing over the reins to OASIS in the summer of 2002, following the release of V3. OASIS (Organization for the Advancement of Structured Information Standards) is an accepted Web- and XML-related standards body. It started its life in 1993 under the name “SGML Open.” This was a consortium of vendors and users committed to developing guidelines for interoperability among products that support the Standard Generalized Markup Language (SGML)—the granddaddy of XML. OASIS (<http://www.oasis-open.org>) adopted its current name in 1998 to reflect its expanded scope of interest, especially in the technical aspects of XML and e-business.

OASIS now bills itself as a not-for-profit, global consortium, which drives the development, convergence, and adoption of e-business standards—UDDI now being one of these. OASIS has more than 2,000 participants representing over 600 organizations and individual members in 100 countries. It also has close ties with the United Nations (UN), and together they jointly sponsor ebXML (Electronic Business using XML)—one of the first, and as yet one of the most credible, initiatives, alongside Web

services, to promulgate the use of XML by enterprise applications.

OASIS sets out to produce respected and popular worldwide standards for security, Web services, XML conformance, business transactions, electronic publishing, topic maps, and interoperability within and between marketplaces. It achieves this goal by enabling its wide and well-represented membership to set the OASIS technical agenda. Cognizant of how heavy bureaucracy has successfully scuttled many standards efforts (and OSI comes to mind as a good example), OASIS favors a dynamic, lightweight, open process expressly designed to promote industry consensus and unite disparate efforts. Given these credentials, one cannot reasonably argue that OASIS does not have the legitimacy to make UDDI a true, industry standard for the Web and e-business.

When UDDI was first conceived, realizing a UBR was the Holy Grail. UDDI V1, as characterized in [Figure 4.6](#), focused on the needs for implementing a UBR. The initial UDDI.org Consortium, which consisted of around 30 well-known names, in addition to those of IBM and Microsoft, was indeed extremely successful in making sure that a three-node UBR was active and ready to roll when the initial V1 specification was unveiled to the world. Since then, the most profound change that has happened has been the growth in interest in private and semiprivate UDDI registries. Version 3's main thrust has to do with multiregistry-distributed UDDI environments with root and affiliate registries.

[Table 4.1](#) sets out to highlight the evolutionary history of UDDI.

Table 4.1: Evolutionary History of UDDI

UDDI Specification			
	Version 1	Version 2	Version 3
<i>Published:</i>	September 2000	June 2001	July 2002
<i>Primary themes:</i>	Create a framework for a UBR	Support complex organization models that include business units and subdivisions “Internationalization” in the form of permitting businessEntity description in multiple languages Support for more taxonomies	Private and semiprivate registries User-friendly UDDI keys “Operation information” (i.e., subscription service for change notification) Support for digital signatures
<i>Data structures:</i>	1. businessEntity 2. businessService 3. bindingTemplate 4. tModel	1. businessEntity 2. businessService 3. bindingTemplate 4. tModel 5. publisherAssertion	1. businessEntity 2. businessService 3. bindingTemplate 4. tModel 5. publisherAssertion 6. Subscription

<i>Registry keys supported:</i>	uuidKeys	uuidKeys, domainKeys, and derived keys
---------------------------------	----------	--

The bottom line here is that the UDDI standard is now well ahead of UDDI implementations, including the UBR—not to mention overall UDDI adoption. In reality, OASIS has time on its hands in terms of having to do much more on the standard—though, as is today’s wont when it comes to Web services–related specifications, it is unlikely to be too long before we see UDDI V4. However, what is important right now is to ensure that UDDI and Web services become valuable business assets, rather than being technologically impressive paper tigers.

Team LiB

Team LiB

◀ PREVIOUS

NEXT ▶

◀ PREVIOUS

NEXT ▶

4.4 UDDI implementations—UBR and otherwise

UDDI found itself at an interesting and important crossroad in the summer of 2003—exactly a year after its custody was transferred over to OASIS. The UBR, now consisting of the four nodes after NTT joined IBM, Microsoft, and SAP as a node operator in October 2002, continues to be real and freely available. [Figure 4.15](#) shows the front page of NTT’s Japanese-centric UBR node to complete the set of UBR node screen shots included in [Chapter 1](#) and this chapter.

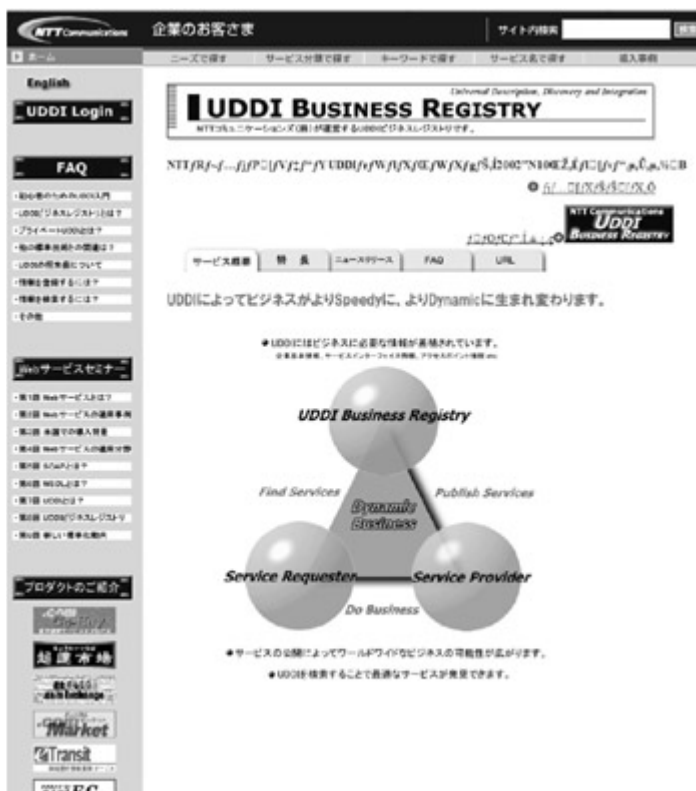


Figure 4.15: NTT Communication’s Japanese-centric UBR node.

Despite the no-cost, no-holds-barred, worldwide availability, the UBR’s popularity, reputé, and applicability are, however, still open to question. As with all things Web services–related, UDDI has not yet even come close to living up to its once-lofty expectations. On the other hand, the inclusion of a full-

function, V2-compliant UDDI service as a standard feature within Microsoft's now flagship Windows Server 2003, which started shipping in June 2003, coupled with Visual Studio .NET's direct support for UDDI search-and-publish functions, gives UDDI a whole new breadth of exposure.

In addition, the availability of Java-based, multiplatform enterprise UDDI servers from the likes of IBM and BEA, as adjuncts to their highly popular application server offerings, gives UDDI even further credibility with application developers. The Java versions, moreover, enable UDDI implementations on UNIX and even iSeries platforms. The Java- and Windows-based enterprise versions of UDDI will certainly help make UDDI more real, tangible, and accessible vis-à-vis the IT community—hence, why it is safe to claim that UDDI really is at a crossroad at the time of this writing.

Up until V3, the focus of UDDI implementations revolved around the UBR initiative. However, V3's explicit endorsement and support for multiregistry environments changes that in a profound manner. Over the next couple of years, the emphasis is going to shift toward enterprise UDDI registries, both private and semiprivate, and their interaction with the UBR. Consequently, before venturing any further, it is best to characterize the features of the key UDDI implementation types, in tabular form, so as to serve as a framework for the remainder of this section. This is done in [Table 4.2](#).

Table 4.2: Features of Key UDDI Implementation Types

	Public UDDI Registries		Enterprise UDDI Registries	
	UBR Business Registry	UBR Test Registries	Private Registry	Semiprivate, Shared Registry
<i>Purpose:</i>	Global, unrestricted visibility to services available from businesses, organizations, and individuals from around the world	A test-bed version of the UBR provided at some UBR nodes mainly to provide software developers with a means to accurately try out the working of the UDDI APIs	<ol style="list-style-type: none"> 1. Internal catalog, for use by in-house software developers of available (and approved) Web services that may be used in new applications 2. Test bed for use by in-house programmers developing programmatic access to the UBR 3. In the case of large multinational, multidivisional enterprises as a standardized, in-house directory, categorized by taxonomies, of all available services—Web or otherwise 	<ol style="list-style-type: none"> 1. List of Web services approved for use in b2b e-business applications being developed by or for use by members of this group of “partners” 2. Standardized list of all services, categorized by taxonomies, being offered by the various members that make up this “partnership”
<i>Accessed via:</i>	Internet		Intranet	Extranet

<i>Behind firewalls:</i>	No		Yes	
<i>Cost:</i>	No charge at present		UDDI server functionality typically included with a “larger” server product	
<i>Administration:</i>	UBR node operators		Private, in-house	Collaborative
<i>Likely to be multinode:</i>	Yes	No	No	Yes

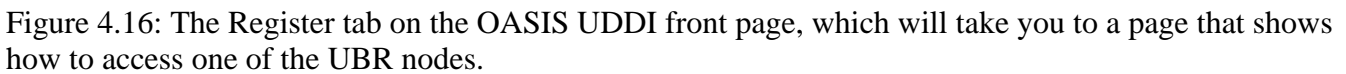
4.4.1 The UDDI Business Register

The UBR is the public face of UDDI. In reality it was to be the be-all and end-all of what UDDI was all about. Though a UBR, per the original expectations, has now been in continuous operation for nearly 3 years, on a no-charge, open-to-all basis, it is fair to say that the UBR is still a little-used novelty—only appreciated by a select, Web services–centric cognoscenti. It is still far from being the oracle for e-business-related services that it was supposed to be.

In its defense, it is only fair to note that following the dot.com debacle, the spread of e-business slowed down dramatically in mid-2001—just as UDDI and UBR were trying to gain momentum. Then there were 9/11 and the resultant economic downturn, particularly severe in the high-tech sectors—as well as a huge, overall increase in cautiousness related to all security- and privacy-related issues. The ongoing epidemic of viruses, Internet attacks, relentless spam, and security exposures on Windows platforms justifiably fuels this “under-siege” mind-set.

Against this background, the UBR was and still is an unknown. Businesses are leery about disclosing too much information about themselves to strangers. They especially do not want to publicize too many electronic links into their organization. The original “free-world” climate within which the UBR was conceived has changed—irrevocably!

This security-inspired, technological “xenophobia,” however, is not the only reason why the UBR has not flourished. The UBR is bereft of leadership. OASIS, true to its charter as a standards body, wishes to stay aloof from UDDI implementations—even if it is the egalitarian, “not-for-gain” UBR. There is no explicit reference to UBR on OASIS’s UDDI-related front page. Instead, there is a misleading and obscure reference on the top navigation bar that says “Register”—with an exclamation mark in front of it, as shown in [Figure 4.16](#). Clicking on that then takes you to a page that is headed “Register.” If you are just looking to sample and evaluate the UBR, this process is confusing and intimidating to say the least.



The URLs for the various UBR nodes (e.g., uddi.microsoft.com, uddi.ibm.com, and uddi.sap.com) are another problem. The “.com” suffix detracts from the service-oriented theme that the UBR should project. Though one of the express goals of the UBR is to promote e-business, the “.com” connotations—especially when juxtapositioned against names such as Microsoft, IBM, and SAP—can understandably create some level of paranoia about how information maintained in the UBR may get exploited in the future. Then, as if to add insult to injury, typing in uddi.ntt.com results in a “page not found” error. Instead, to get the NTT node, you have to enter <http://www.ntt.com/uddi>.

Ironically, however, another factor impacting the UBR is the lack of UDDI-related commercialization. To date, nobody, including IBM, Microsoft, or SAP, has made much money on Web services, UDDI, and, by implication, the UBR. UDDI- and UBR-related ROI have to be in the negative territory. This explains why the UBR, at present, is not as slick as one would expect—particularly in comparison to large search engines such as Google. First, a year after V3 was released, all of the UBR nodes were still at V2 levels. Given that it is V3 that permits cooperative coexistence of multiregistry environments, the lack of V3 support also has a dampening effect on enterprise-level implementations.

12/18/2009

global repository of all available services. One hopes that as the UDDI message becomes better known within enterprises, thanks to the enterprise UDI implementations, the popularity of the UBR will increase—even if it is, to begin with, just to enumerate traditional services not associated with Web services. In the interim, the primary pros and cons of the UBR are summarized in [Table 4.3](#).

Table 4.3: Pros and Cons of the UBR

Pros	Cons
<ol style="list-style-type: none"> 1. Standards based, open service 2. No charge (at present) and available to all without discrimination 3. Global, Web browser accessible service 4. Published API-based programmatic access 5. Powerful, extensible architecture that permits complex queries 6. Multinode implementation fosters redundancy 	<ol style="list-style-type: none"> 1. Still lacks visibility, not to mention credibility 2. No checks on the validity, accuracy, or completeness of the entries 3. Lacks consistency across the nodes 4. Predominantly English oriented, with the NTT node being Japanese 5. No perceived driving force 6. Unintentionally succeeds in projecting a highly commercialized “sinister” image, though it is essentially a not-for-profit service 7. Has yet to cogently quell security and integrity issues that businesses may have

4.4.2 UDDI enterprise registries

The growing interest in private and semiprivate UDDI registries echoes a fundamental ground shift that has occurred in the Web services world. When originally conceived, Web services, as testified by the all-important word “Web” in the name, were going to be all about standards-based software functionality available over the Web. Web services were the software methodology sector of the Web-based, “the whole world is the market,” free-trade zone that transcended geographic, political, currency, cultural, and economic boundaries. Well, this utopian dream of application developers around the world, gainfully collaborating with each other over the Web, is not going to happen as quickly and as easily as originally envisaged.

There is too much mistrust and foreboding in the post-9/11 business world. In addition, the constant state of high alert in which businesses now have to operate when it comes to IT-related attacks, thanks to the never-ending news about various security flaws in heavily used software, further exacerbates all concerns regarding the security, privacy, and protection of all corporate assets. Consequently, more and more enterprises are looking at Web services as a technology they want to use either strictly in-house or just with trusted partners. Rather than “Web” services, it is now becoming [“intranet”](#) services or [“extranet”](#) services.

This situation is responsible for the increased interest in enterprise registry. Enterprise registry enables enterprises to faithfully recreate the entire Web services operational model—albeit now purely on an intranet or extranet basis. The in-house corporate phone book analogy immediately and obviously comes to mind. With private registries, enterprises can maintain their own listings of services, per the UDDI

model—whether for Web services or otherwise. They will have total control of the access, administration, and security aspects of these in-house registries. They can be maintained on a strictly need-to-know basis.

Only those who are actively involved with particular projects need to even know that an enterprise has an in-house UDDI Registry. Just as with information listed in a corporate phone book, most enterprises will not want the contents of their private UDDI registries known to outsiders. As with other sensitive corporate information, there will be rules, policies, and procedures as to how information contained in a private UDDI Registry is treated.

Private and semiprivate UDDI registries can be a major boon for in-house software developers—particularly as they transition toward a Web services–centric software development model. These registries provide them with a standard mechanism for cataloging the new Web services they develop. No longer will they have to rely on e-mails, departmental meetings, and word of mouth to disseminate, plus acquire, information about the in-house Web services now offered. They can gainfully use the in-house registry—in exactly the same way that UBR was originally intended to be used as the self-advertising mechanism for Web services.

Private and semiprivate registries can also be the staging posts for automated, programmatic, Publish API set–based entries into the UBR, once the UBR nodes, as well as the enterprise implementations, are all at V3. As previously mentioned, IBM, Microsoft, and SAP offer test registries alongside their business registries for programmers who wish to test their software against a real, live UDDI implementation. However, using one of these test registries still means going outside the firewall, so the enterprise implementations, especially those that can be realized to be extremely economical using the bundled-in UDDI server inside Windows 2003, will satisfy an important need. Programmers like to try out their software. Now they have a real emulation that they can beat on, behind the privacy afforded by firewalls, to their hearts' content, before they have to trot their efforts out into the glare of the Web.

Private and semiprivate registries also allow programmers to have the option of developing truly dynamic, “don't bind until run time” applications based on Web services. Remember that the whole Web services model is contingent on invoking and binding with the necessary Web services modules at run time. Web services are an RPC mechanism. UDDI, through the bindingTemplate entities, can maintain all necessary invocation parameters for a Web service. Given this framework and the invocation pattern inquiry scheme supported by the UDDI Inquiry API set, it is possible to write new applications that only invoke Web services via their UDDI entry. This means that the actual Web service invoked could change as new Web services are added to a UDDI Registry. With enterprise registries, developers, when applicable, could exploit this highly dynamic Web service invocation model.

The bottom line here is that enterprise UDDI registries, as with employees-only intranet or partners-only extranet portals, are here to stay. In today's security-obsessed corporate culture, they fulfill an important need for providing full UBR compliance within the privacy and security afforded by a firewall. The sponsors of UDDI recognize this need—hence, the support for multiregistry environments in V3. All that is required now is for the UBR as well as enterprise implementations to be based on V3 so that the interoperability envisaged by V3 can be a reality.



4.5 Security and integrity in relation to UDDI implementations

As with all things Web-related, post-9/11 security, as already discussed in the preceding sections, has hampered the spread of UDDI and the UBR. However, when it comes to UDDI and the UBR, one has to be very careful and judicious as to what one means by security—and the scope of potential security exposures in this context. The first distinction that has to be made is between enterprise implementations and the UBR. Enterprise implementations, whether private or semiprivate, are meant to be closed user group affairs, which should only be accessible to authenticated and authorized users. The corporate phone book analogy has already been used.

In the case of private registries, system administrators should implement all appropriate access control mechanisms to ensure that only those with a need to know have the necessary rights to search, populate, or update the registry. This goes without saying. A private registry becomes just another sensitive corporate IT asset, and the type of corporation that would need a private registry will already have mature security policies about such IT assets. Obviously the one thing that you do not want is unauthorized users being able to snoop around or modify the contents of a private registry.

With semiprivate extranet registries, the issue of security becomes more difficult—but not insurmountable. First, having an extranet registry that is shared among trusted partners does not preclude you from also maintaining an intranet-only private registry. Thus, you can have multiple levels of security and access controls. Within this structure, you would only publish select and carefully vetted information in the semiprivate registry.

Companies looking at semiprivate UDDI implementations are most likely to already have some level of experience when it comes to b2b e-business. There is a high possibility that they already operate a partners-specific extranet portal. As a result, such companies will already know how to share and at the same time restrict information between partners using authentication, personalization, and access controls. All of these will come into play when implementing a successful and secure extranet UDDI registry.

The security issues surrounding the UBR are, however, totally different. One can even plausibly argue that when it comes to the UBR, what is at stake is not so much security—but integrity. What is important to remember here is that the UBR, by definition and intent, is meant to be a public repository. Hence, to read that the UBR node operators go to great lengths to encrypt the information maintained by a UBR node is incongruous. That is akin, if you think about it, to a public library having all of its books behind lock and key. The information published to the UBR is meant to be in the public domain.


What is key, however, is to maintain the integrity of the information in the UBR, and this is a challenge. Remember the bad old days when people poached domain names that should by rights belong to others? Well, a similar problem is still there with the UBR. As yet, there is no controlling authority that truly validates users who wish to publish new entries in the UBR. Consequently, an unscrupulous but resourceful “joker” could hijack the UBR entries for genuine businesses if those businesses have not already registered with the UBR. Thus, frivolous and inaccurate information is one of the big dangers facing the UBR and its users. To this end, recall that IBM lists 170 categories of services versus the one stated by Microsoft. Is one or both of these entries a joke? That is the problem. It is hard to tell.

Yes, the UBR nodes insist that you register with them before you can publish an entry. In the case of IBM, this involves selecting a previously unused user ID and an appropriate password and providing a set of contact information, with a valid e-mail address being sacrosanct. Your registration is only activated when you invoke a link to send to the e-mail address provided. This e-mail-based validation approach is not new and has been widely used in the last 5 years or so as a minimally acceptable, baseline method.

The rationale here is that the service provider (i.e., the UBR node operator) has at least an e-mail address that once worked and that points to the user trying to register. Suffice to say that this e-mail-based validation is really not worth the electrons used to execute it! A wily hacker will not have any problems opening numerous hard-to-track e-mail accounts. Microsoft insists that registration to use its UBR node has to be made via its now severely tarnished and undermined Passport service. This alone, as discussed in [Chapter 3](#), may be enough to convince cynics that the integrity of the UBR is already compromised!

This is where digital certificates and digital signatures come in. UDDI V3 supports these mechanisms, but the exact use of these is left as policy decisions to the individual node operators. It now seems inevitable that down the road, all the UBR node operators will insist upon third-party certification, à la digital certificates, in order for one to register as a bona fide UBR publisher. Until this happens, there will always be concerns about the integrity of the information available in the UBR.

Once an entry is published, only the user who published that entry can modify or delete it. This is good and reassuring, assuming that the original user who did the publishing was legitimate and was who he or she actually claimed to be. The original publisher, through the custody transfer API set, also has the ability to assign ownership to another registered user—the key here, though, being that the original publisher has all the initial rights. This is why it is so imperative to ensure, via digital certificates, that the original publisher is not a charlatan. Once there is a validated publisher, a UBR node's policies could further dictate that it will only accept digitally signed entries from that user. With such policies one can put stock in the integrity of the information maintained in the UBR. During the transition to a digitally verified UBR structure, it will be possible, albeit with V3 implementations, for users to request that the searches they conduct on the UBR be restricted to those entries that have been digitally verified as to their authenticity.

Team LiBTeam LiB◀ PREVIOUS NEXT ▶◀ PREVIOUS NEXT ▶

4.6 Q&A: A time to recap and reflect

Answers

A: Universal Description, Discovery, and Integration is a scheme for Web-based electronic directories that contain detailed information about businesses, the services they provide, and the means for utilizing these services. A UDDI directory, referred to as a registry, is meant to be platform independent and can be readily accessible via a Web browser-based GUI or by applications via published APIs. Its goal is to ensure that enterprises and individuals can quickly, easily, and dynamically locate and make use of services, in particular Web services, that are of interest to them.

A: Yes, it is a de facto industry standard under the auspices of OASIS (<http://www.oasis-open.org>). OASIS, which was formed as a consortium in 1993 to promote SGML, is now an accepted standards body for XML and e-business-related initiatives. Thus, it makes sense for UDDI to be under the stewardship of OASIS—which is a not-for-profit agency with more than 2,000 participants representing over 600 organizations and individual members in 100 countries. In addition, UDDI is based entirely on XML and SOAP, which in turn are acknowledged industry standards.

A: Yes. Unlike XML or SOAP, UDDI is not a prerequisite “protocol” that has to be used by a Web service in order for it to be deemed a Web service. UDDI is essentially an optional adjunct to publicize the availability, characteristics, and activation requirements of Web services. It is possible to write applications (and Web services) that can dynamically locate and invoke Web services by programmatically interrogating a UDDI Registry. However, this is by no means mandatory; it is an option. Most applications developers are unlikely to opt for such a dynamic approach, given the variables and uncertainty of this type of on-the-fly run-time binding. Though not a prerequisite per se, UDDI is the accepted means for realizing the self-advertising mandate for Web services—and as such should be considered as an integral part of a Web services landscape.

A: Despite what would appear to be a common goal when it comes to describing the characteristics of Web services, there is no prespecified relationship between UDDI and WSDL. In contrast to SOAP, WSDL is not one of the prerequisite standards used by UDDI. They do, however, complement each other. One of UDDI’s express missions is to publicize the technical requirements and invocation criteria for Web services. WSDL happens to be the preferred and accepted way to describe the interface of a Web service. Thus, there is an obvious and natural role that WSDL can play vis-à-vis UDDI. The tModel information for a Web service within a UDDI Registry could thus point to a WSDL description of that Web service, as can the bindingTemplate.

A: A UDDI Registry is an instance of one complete set of UDDI-related data about businesses, the services they provide, and technical details of these services. A UDDI Registry is made up of one or more UDI nodes, where a UDDI node is a software system that supports at least one of the UDDI API sets. A given UDDI node can, at any one time, only be a member of just one UDDI Registry. In the case of a multinode registry, all the nodes are expected to work collaboratively with each other, representing one unified view of the data managed by that registry.

A: No. UDDI is not built upon any of the popular directory schemes. UDDI is meant to be a high-level, “logical” overlay architecture that deals with data structures, in the form of XML schema, and SOAP-based messages, invoked from APIs, to manage and manipulate these data structures. The UDDI specification, now in its third version, does not specify or even suggest how one should go about realizing these data structures or the SOAP messaging. Those are left as individual implementation decisions. Thus, there is no formal relationship for the Lightweight Directory Access Protocol (LDAP) or even X.500. There are, however, specifications available on the Web as to how a UDDI node could be implemented on top of LDAP or X.500. At present, however, many of the UDDI implementations are realized using popular relational database systems such as IBM’s DB2, though BEA, for one, offers an LDAP-based UDDI implementation.

A: The Universal Business Registry, also referred to as the UDDI Business Registry, is a global, public UDDI service available on the Web that is open to all—currently on a no-charge basis. It has been in operation since late 2000. The UBR was initially operated by Ariba, IBM, and Microsoft. Today, the UBR consists of four UBR nodes, working collaboratively, with these nodes being run by IBM, Microsoft, NTT Communications, and SAP. You can locate the UBR by going to

<http://www.uddi.org> and clicking on the “Register” tab at the top of the page.

A: At present there is no charge for using the UBR—whether to publish information or to search for information. The UBR nodes, however, require those who wish to publish information in the UBR to be registered with them beforehand so that they can be authenticated. Anybody can freely search the UBR without being registered. Down the road it is highly likely that the UBR nodes will require potential publishers to be digitally authenticated and certified before they can apply for registration. There is likely to be a charge for this digital certification.

A: Yes. Version 3 of the UDDI specification, which was published in July 2002, endorses the concept of multiregistry environments consisting of a root registry (typically the UBR) and multiple affiliate registries. Today, IBM, Microsoft, BEA, Sun, and others offer products for implementing so-called enterprise UDDI registries. An enterprise UDDI Registry could be private (i.e., restricted for use by one company) or semiprivate (i.e., shared between trusted partners). Microsoft’s flagship Windows Server 2003, which started shipping in June 2003, includes a full-function UDDI Registry capability as a built-in feature.

A: No. Though UDDI and the UBR were conceived to promote e-business, and in particular Web services, neither the UDDI specification nor the UBR insists that the only services that can be included in a UDDI Registry are Web services. In reality the UBR is extremely generic and flexible. One can register any type of business, organization, or even an individual, as well as any arbitrary type of service.

Team LiB

◀ PREVIOUS NEXT ▶