The role of private UDDI nodes in Web services, Part 1: Six species of UDDI

Steve Graham (sggraham@us.ibm.com)
Web Services Architect, IBM Emerging Internet Technologies
May 2001

Steve Graham introduces the concepts behind Web services discovery and gives a brief overview of UDDI (Universal Description Discovery and Integration). He examines six variants of UDDI registries, highlighting the role each of these plays in a service-oriented architecture.

(Participate in the discussion forum on this article by clicking **Discuss** at the top or bottom of the article.)

Contents:

UDDI
Six species of UDDI
The UDDI operator
The e-Marketplace UDDI
The portal UDDI
The partner catalog UDDI
The internal enterprise application integration UDDI
The test bed UDDI
Conclusion
Resources
About the author
Rate this article

Related dW content:

UDDI4J: Matchmaking for Web services

In service-oriented architectures, service descriptions and metadata play a central role in maintaining a loose coupling between service requestors and service providers. The service description, published by the service provider, allows service requestors to bind to the service provider. The service requestor obtains service descriptions through a variety of techniques, from the simple "e-mail me the service description" approach and the ever popular sneaker-net approach, to techniques such as Microsoft's DISCO and sophisticated service registries like the Universal Description, Discovery and Integration (UDDI), which is what I am going to examine here.

UDDI

UDDI defines four basic data elements within the data model in version 1.0: businessEntity (modeling business information), businessService (high level service description), tModel (modeling a technology type or service type), and bindingTemplate (mapping between businessService and tModels). I won't discuss the intricacies behind these elements, so if you need more basic information on UDDI, please visit the UDDI web site before continuing (see Resources).

The set of operator nodes known as the UDDI business registry, or *UDDI operator cloud*, implies a particular programming model characterized by design-time service discovery. We need design-time discovery since it is often not feasible to implement dynamic discovery at run-time due to overwhelming complexity.

However, the *just-in-time integration* value proposition of the IBM Web Services Initiative allows organizations to provide dynamic discovery and binding of Web services at run time. To do this, API characteristics and other non-functional requirements are specified as business policies at design time. This flexibility has important characteristics for loosely-coupled enterprise application integration, both within and between organizations.

The role of the UDDI cloud to support a dynamic style of Web services binding is currently limited. However, the UDDI API and data model standard can still play a role in a service-oriented architecture. The notion of a private or non-operator UDDI node is critical to the emergence of a dynamic style of a service-oriented architecture.

Six species of UDDI
UDDI is really two things: 1. the *UDDI cloud of operator nodes*, an Internet-wide repository (made up of white, yellow, and green pages) for Web services metadata; and 2. an *API and data model* standard for a Web services metadata repository. The former hosts the data while the latter provides a means to access it.

Currently, the most important form of UDDI is the UDDI operator cloud. The UDDI operator cloud is a collection of UDDI nodes coordinated by an operator's agreement to provide uniform access to Web services metadata from any of the operator nodes. If Web services metadata is entered in one operator node, it can, with some reasonable time delay, be retrieved from any of the other operator nodes. The operator nodes share the data using a well defined replication scheme.

The structure of the UDDI specification allows the possibility of private UDDI nodes. A private UDDI node can implement all of the UDDI APIs as defined by the UDDI v1.0 specification, but it does not participate in the replication scheme defined by the UDDI operator’s agreement.

I have identified six variants, or what I call species, of UDDI:

1. UDDI Operator node
2. e-marketplace UDDI
3. Portal UDDI
4. Partner catalog UDDI (also known as Vetted partners or rolodex-like UDDI)
5. Internal Enterprise Application Integration UDDI
6. Test Bed UDDI.

All but the first species, UDDI operator node, are considered private or non-operator nodes. Because of the data volumes, the breadth of industry, geography, products covered, and the requirement of adherence to the operator agreement, operator nodes are limited in the degree of additional or variant behavior they can provide. Private nodes are not under the same restrictions.

I will now briefly discuss each type of UDDI node.

The UDDI operator
This is what most folks think of when they consider UDDI. These UDDI nodes form the operator cloud facility that can be accessed from www.uddi.org (see Resources). The prime example of this is the UDDI Business registry currently replicated between IBM, Microsoft, and soon HP (see Resources). Ariba, formerly an operator, has just announced that it is shifting to the role of a registrar of Web services.

The e-marketplace UDDI
This private UDDI node is hosted by an e-marketplace, a standards body, or a consortium of organizations that participate and compete in the industry. In this UDDI node, all the Inquiry APIs (that is, *publish* and *find*) are deployed for access over the Internet by any of the member organizations.

The entries in this species of UDDI relate to a particular industry or narrow range of related industries. Such a membership process allows the entries to be pre-filtered to include only legitimate businesses participating in the industry. The membership process can also restrict who is allowed to invoke find operations against the UDDI node. For example, the steel manufacturer’s association may choose to host a UDDI node, allowing its members to register their business and Web services.

An e-marketplace UDDI is a target-rich environment for finding Web services metadata for doing business within a particular e-marketplace or industry. The e-marketplace UDDI would be the logical place to find industry-specific custom taxonomies (standard product coding hierarchies, specializations of North American Industry Classification System [NAICS] categories, and so on) as well as standard Web service interface definitions (tModels) for common business processes in the industry. For example, vendors would be able to determine which types of purchase order placements are typically made in the steel industry by examining the tModels that are contained in the e-marketplace’s UDDI. As industries agree on standard Web services interface definitions, Web services adoption will accelerate.

This style of private UDDI node allows an e-marketplace to provide value-adds like quality of service monitoring on the partner’s Web services response times and Better Business Bureau style industry self-monitoring of its members' Web services business practices.

Finds in serious machine-to-machine (M2M) B2B will happen in the e-marketplace UDDI node. A buyer may issue a find against an e-marketplace UDDI node in preparation for the issuing of a request for quotes (RFQ) to various suppliers within an industry. A simple find against this UDDI node reveals all the suppliers able to receive a RFQ via a Web service invocation. These e-marketplaces will want to move into the Web services arena to avoid being dis-intermediated by the UDDI operator cloud.

The portal UDDI
The transactional Web refers to use of the Internet for M2M B2B. Just as a company has a Web presence on the eyeball Web (http://www.acme.com), so too will it have presence on the transactional Web (http://www.acme.com/uddi). I call this transactional Web presence a *portal UDDI*.

The portal UDDI typically sits on the company’s firewall in its demilitarized zone and is a private node that contains only metadata related to that company’s Web services. So the entries in www.acme.com/uddi reflect the metadata for Web services that acme.com wishes to provide to external partners. Clearly it is in the company’s interests to keep the find APIs accessible from the Internet, although the publish APIs will be removed, restricting publish to internal processes only.

When acme.com wants to deploy a Web service, it publishes the Web service description to its portal UDDI. Other facilities, described in the next installment of this article, are then automatically invoked to make sure the new Web services metadata is properly reflected in other UDDI nodes, such as the operator cloud, various e-marketplaces, and key partners catalogs.

The portal UDDI gives a company ultimate control over how metadata describing its Web services is used. For example, a company can restrict find access. It can also monitor and manage the number of lookups being made against its data and potentially get information about who the interested parties are.

With respect to the UDDI operator cloud, the URL for acme.com’s transactional Web presence UDDI node can be used as the discoveryURL in the businessEntity record for acme.com.

The partner catalog UDDI
This private UDDI node sits behind the firewall and finds and binds are made against it. It also only contains Web service description metadata published by trusted business partners (that is, those organizations with which acme.com has formal agreements/relationships). The publish and find APIs are not available over the Internet, so only internal applications have access to all APIs.

The partner catalog allows an organization to build applications in a service-oriented way, taking advantage of dynamic binding against Web services through a Eeb services interface established at design-time. The API to the Web service, as embodied by the Web services interface definition (for example, a conventional use of WSDL using a tModel), is fixed at design-time. But other characteristics, such as transport, security stack, and so on can be dynamically accommodated at run-time. Applications are coded to accept any Web service that matches a find executed against this UDDI. Because the UDDI node contains only approved business partners, this dynamic binding does not raise the risk of dealing with an unknown service provider.

Consider the following scenario. Buyer.com wants to do M2M e-business using Web services techniques in order to reduce transaction costs for supply-chain tasks, such as supply-reordering, but they don't want to be locked into dealing with a particular vendor. The management at Buyer.com agrees to do business with acme.com. Buyer.com’s IT staff can examine the UDDI entries for acme.com (either from the operator cloud, an e-marketplace UDDI, or acme.com’s portal UDDI) and copy these entries into its own Web services partner catalog UDDI. Buyer.com repeats this process for all vendors it wishes to contract with. Now as the list of suppliers changes over time as business relationships are formed and dissolved, the changes in the entries within Buyer.com’s partner catalog UDDI node will be updated.

Proxies can be generated from WSDL standards and applications written to carry out finds against the partner catalog UDDI node. The applications do not need to be recoded to cope with the changing list of approved partners. The application uses the UDDI entries to determine the Web services available, it chooses one or more of the best Web services to invoke, based on a given business policy, and then it invokes the Web service.

To make this dynamic binding application support complete, the partner catalog UDDI node can restrict the publish function to allow only entries under a certain criteria (for example, only businessServices associated with a WSDL based tModel or only businessServices which support a well-known industry standard purchase order tModel). This allows the administrator to guarantee the shape of entries that are placed within the UDDI node and, therefore, what applications can expect in response to find operations.

The partner catalog UDDI node also supports value-added services like find businessService based on tModel key, or find businessService based on WSDL type, regardless of the business that supports it. These additional APIs support the scenarios described above.

The internal enterprise application integration UDDI
This resembles the partner catalog species, except that the entries are for Web services provided by other departments or groups within an organization. The major distinction of this species is the potential for common administrative domain that can dictate standards (for example, the tModels used, common use of WSDL portTypes, and so on). This allows the UDDI node to operate with different publish restrictions than those suggested for the partner catalog UDDI. For example, the node could restrict the publication of new tModels and thereby restrict publishing of businessServices and bindingTemplates to accept only entries associated with a fixed set of tModels. The fixed set of tModels corresponds to the technology standards chosen by the common administrative domain.

Of course, this species exists completely hidden behind an organization’s firewall. Publish and find operations would be restricted to applications within an organization.

The test bed UDDI
This type of private UDDI node tests applications. The testing can be for both requestor applications and provider Web services.

If acme.com provides Web service access to their purchase-order placement-system, acme.com uses this type of UDDI node to make sure of two things: 1. that the UDDI entries for their Web service are accurate and 2. that applications can use the Web service information to generate proxies from the UDDI entry to access the Web service.

If acme.com builds applications to consume Web services, it tests the application's ability to cope with different variants of Web service description with entries from the test bed UDDI. Acme.com will clearly want to run some trials against any new UDDI entry discovered in the operator cloud, or an e-marketplace UDDI or partner’s transactional Web presence UDDI. Entries will typically be copied here first. A battery of tests are then run to make sure the application can use the information found in the entry. And then, only after testing, the entry will be promoted to the vetted partner's UDDI, or the enterprise application integration UDDI node.

Conclusion
We have briefly examined the discovery role played by UDDI within a service-oriented architecture and enumerated six species of UDDI, each supporting different uses of a service-oriented architecture. In the next installment of this article, I will contrast the programming models that use private and operator UDDI nodes, and review requirements for functionality to make private UDDI nodes easier to use.

Resources
- (Participate in the discussion forum on this article by clicking **Discuss** at the top or bottom of the article.)
- Find out more about Microsoft's DISCO.
- Check out the official UDDI site.

- Take a look at the collection of UDDI resources.
- Take the IBM UDDI Test Registry for a trial run.
- Go to the official IBM UDDI registry to post Web services and publish them to global audiences.
- Check out this UDDI Browser.
- An open-source Java-based implementation of UDDI is available for review.
- Take a look at the UDDI4j project hosted on developerWorks.
- Find out about the UDDI day that happened at the XML DevCon.
- Read the XML cover pages.
- Take a look at the Microsoft UDDI pages.

About the author

Steve Graham is an architect in the Emerging Technologies division of IBM Software Group. He has spent the last several years working on service-oriented architectures, most recently as part of the IBM Web Services Initiative. Prior to this, Steve worked as a technologist and consultant working with various emerging technologies such as Java and XML, and before that he was an architect and consultant with the IBM Smalltalk consulting organization.

Before joining IBM, Steve was a developer with Sybase, a consultant, and a faculty member in the Department of Computer Science at the University of Waterloo. Steve holds a BMath and MMAth in computer science from The University of Waterloo.You can reach him at sggraham@us.ibm.com.

Discuss e-mail it!

What do you think of this article?

( ) Killer! (5) ( ) Good stuff (4) ( ) So-so; not bad (3) ( ) Needs work (2) ( ) Lame! (1)

Comments?

Submit feedback

Privacy | Legal | Contact