Advertising a Web Service

In today's sober business environment, whiz-bang technology doesn't capture the CEO's imagination the way that it did during the dot-com boom of the 1990s. However, this does not mean that they are not open to suggestions on how to improve the bottom line; pragmatic and incremental approaches mesh with the more conservative times.

NEW TERM In the world of Web services, strategies that map to the goals stated previously are being funded, whereas those that promise a "revolution" or a "paradigm shift" are falling on deaf ears. One of these paradigm-shift technologies is the concept of a *universal registry* of Web services.

The promise of the universal registry has taken quite a hit in the last 18 months. All the talk about loosely coupled pieces of software that find each other, connect automatically, and exchange data sounds like dot-com happy talk when you hear it said out loud. The idea of the universal public registry is not dead; it has just gone into hiding until both the technology and the public mood is more conducive to it.

This doesn't mean that the ideas are not valid, or that this directory will not exist someday; it is just that it will appear incrementally in certain low-risk, high-reward situations. Over time, as both technical and business leaders gain confidence in the approach, it will become a groundswell that will have far-reaching effects. In this hour, we will look at the issue of communicating the existence of your Web service to others who might be interested in it. We will also examine the opposite situation in which you learn about the services offered by other organizations.

Not all registries are public, however. Some registries can exist within the firewall of one company or in an area where only a few trading partners access it. We will look at these also.

In this hour, you will learn about

- The need to advertise a Web service
- The value of a registry
- The UDDI standard
- The UDDI architecture
- Programming the UDDI
- Creating private registries
- Using a UDDI registry

The Need to Advertise a Web Service

Some business goals are steady in spite of the business climate. Every company president believes that he pays too much for raw materials and services, has too few customers, and needs to expand into new product lines. Any strategy that maps to lowering costs, finding new customers, or identifying new sources of revenue will get a fair hearing. Lowering costs is the reason that *Enterprise Application Integration (EAI)*, supply chain management, and *Enterprise Resource Planning (ERP)* projects are so popular today. Web services projects that are mapped to making these activities easier are providing valuable experience and early success stories.

Whenever new technology appears, its first impact is often to provide an incremental improvement in old processes. The first use of the railroad was to lower the cost of transporting goods over a distance. As important as this was, it was surpassed by the ability to send salesmen over a distance to find new customers. The increase in sales that were generated by their activities caused factories to run day and night to meet the new demand.

As we explained in Hour 2, "Advantages of Web Services," they can be effective in lowering the cost of doing business by making it easier to get data from one computer to another. This is analogous to lowering the cost of shipping freight. The promise of getting new customers or developing new products requires more than SOAP and WSDL; it requires a registry. This registry is analogous to the traveling salesman in the age of railroads.

A registry is essentially a database of services. An analogy to today's computing environment can be found in the early days of the Internet. TCP/IP, Sockets, and HTML were in place and functioning properly in 1994 when a couple of college students at Stanford decided to create a list of Web sites that they found and store them along with keywords that allowed users to perform searches. The result was called Yahoo!, and the rest is history. Before Yahoo!, a person had to know the address of a Web site to access it, or use a search engine to locate the site. Books were published that listed some of these sites. Since the introduction of Yahoo! and its numerous competitors, most users connect to sites that they did not know existed seconds before when they entered keywords in the search engine.

A search engine is essentially a set of words linked to a URL. A registry of Web services will be much more detailed because of the difference in complexity—a Web service is far more complex than an ordinary Web site. The analogy holds though because a registry could allow customers (who didn't even know that you existed) to do business with you. This is the promise of the universal public registry.

Think of how most businesses get new customers today. A salesman learns about a company from reading magazines, talking to friends, or searching the Web. He contacts that business by telephone or letter and tries to set up an appointment with a decision maker. This potential customer struggles to understand what value your salesman is proposing to bring. If the salesman does a good job, you get an initial order from a new customer.

This model might never change for selling certain products such as aircraft engines, where each item is very costly and the potential customers are few. The vast majority of products and services for sale don't fit this pattern. For these products, there are customers who would order them at a good price if they only knew where to find them. Conversely, you are likely buying raw materials today from vendors who are not the best ones in the world. The same products are probably being offered for sale by vendors that you don't know about at a better price and with better terms.

Unless you have penetrated every corner of your geographic area, be it local or worldwide, you don't know about some very attractive trading partners and they don't know about you.

The Purpose of a Registry

You could solve the problem of finding new trading partners through traditional advertising. Many companies purchase television commercials or print ads to try to reach these new customers. Although this approach works well for products such as cars and personal computers, it is not such a good fit for ERP software vendors and custom metal extrusion shops—business-to-business relationships. The rule in advertising is that you pay for every pair of eyeballs that sees your ad, whether or not they are really potential customers. This is why soap is advertised during the afternoon and beer is pitched during football games. The demographics of the viewers matches the products' primary customers, thereby making the ads more effective.

Trade magazines carry this concept one step further. They tailor their articles to attract a narrow segment of the market. They then sell advertising to clients who want to reach that group of customers.

Vending Registry Entries

A registry is just an evolutionary step down that same path. The idea is that a potential customer would use the registry's search facilities in a similar fashion to the way we use Internet search engines. We type in some description of what we are looking for, and a list of potential vendors appears. You further refine your search until you have narrowed it down to the ones that are a perfect match to what you are looking for. You then make a connection to that service and send them a purchase order. They send a confirmation response and ship the goods to your loading dock.

The original vision was that all this interaction would take place without human intervention. In the near term, it appears that the vast majority of the Web services connections that will be made will involve quite a bit of telephone time between the potential vendor and customer. Figure 11.1 shows this graphically.

Rather than be discouraged by the fact that the process isn't completely automatic, we should be glad that the discovery part of the scenario worked. A new customer now knows about you and is placing orders. As the technology matures and comfort levels with it increase, this diagram will become more streamlined. You can probably remember when everyone was reluctant to enter their credit card numbers over the Internet.

Purchasing-oriented Registry Entries

Entries in the registry are not only made by vendors, but also by customers. Many vendors call on large companies because they purchase such large quantities. Normally, an organization will go through a periodic review, and then select a small number of vendors for each category of items that they purchase. This is done to reduce the amount of time that it takes talking to so many sales people.

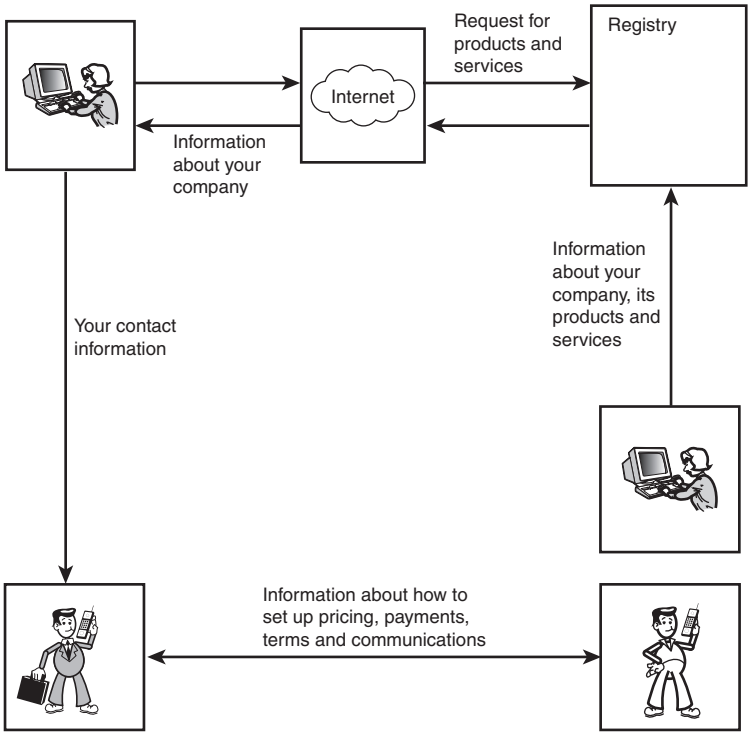

FIGURE 11.1
The registry allows a customer to discover your company.

A company could write a Web service that accepts requests from vendors to compete for future purchases. The vendors could use the registry to find companies who purchase the items that they sell. They could interact with the customer's Web service to obtain approval from the company to submit bids. Whenever the customer gets ready to purchase additional quantities of items, they would send the request for pricing to all the approved vendors of that item. (This request could be sent to a vendor Web service.) Each vendor could then submit a bid based on how desperately they want to make the sale. This price could vary by the supply and demand conditions at each vendor's factories. Figure 11.2 shows this scenario.

Creating a Private Registry

Registries don't have to be public to be useful. They can be very useful behind the firewall of one company. Many companies in the Fortune 500 have dozens of divisions and subsidiaries. Many of these companies solve the same problems over and over at different locations. At times, they try to migrate an application from one part of the company to another, but much duplication remains. Software reuse has proved beneficial at the

system-software level. We all purchase database management systems, networking software, and operating systems from third-party vendors. We have not had as much success in reusing our own applications. Aside from some framework software that companies use, most applications are single use.

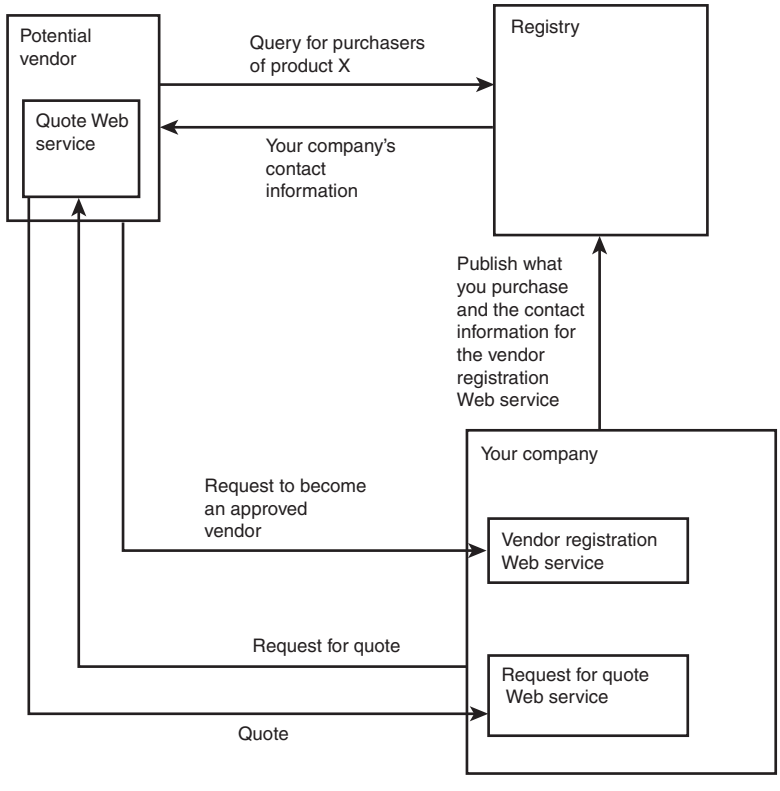

FIGURE 11.2
The registry allows a vendor to discover your company.

For software reuse to work, it needs to bridge operating-system and language barriers. Its interfaces also need to be described precisely. Finally, there has to be a catalog of these applications so that an employee in California can discover that the division based in Florida already has a system that he can use.

The solution to all these requirements is a Web services approach. SOAP provides the operating system and language independence. WSDL provides the precise description of the application, and a private registry provides a way for the engineer in California to find out about the service. Figure 11.3 shows this graphically.

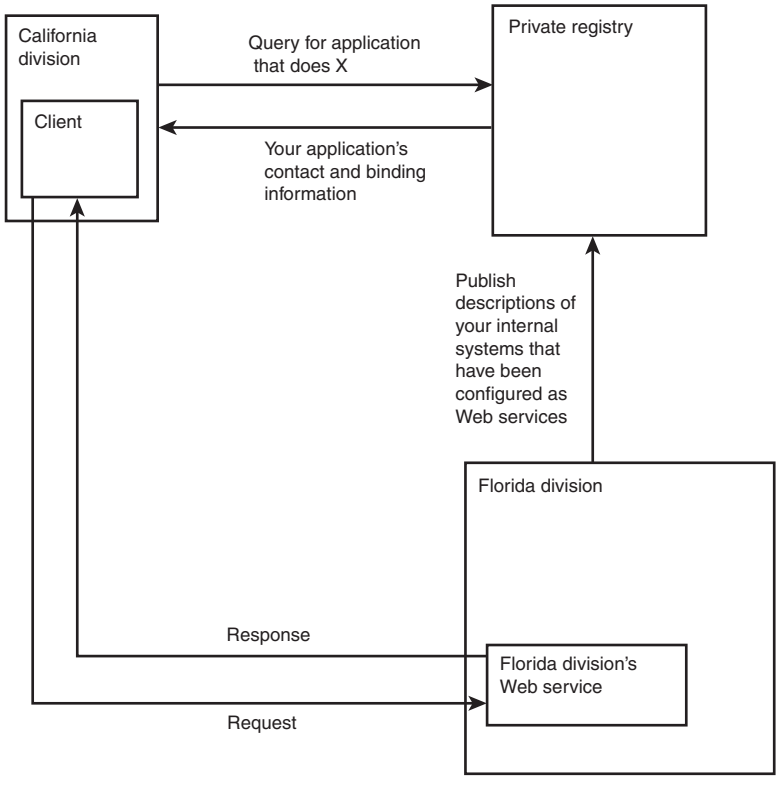

FIGURE 11.3
A private registry facilitates code reuse within your company.

Creating a Semiprivate Registry

NEW TERM One step along the evolutionary path from private to public registries is the hybrid, or *semiprivate registry*. This is a registry that can only be accessed with permission by trading partners or customers. It functions much like a public registry, but it provides its owners with some peace of mind because it is not open to every hacker on the planet—these semiprivate registries sit on a company's intranet and are protected by the company's existing IT infrastructure. Requiring proper authorization permits an organization to expose systems via Web services that are too strategic or sensitive to expose to the public.

A public registration Web service could be used in the vetting process. A company that is interested in becoming a trading partner would use the public registry to locate the "put in an application" Web service. Upon approval, trading partners would be given the keys and passwords needed to access the real Web services exposed by the company.

In addition, information about test versions of the company's Web services could be placed in the registry. These versions could be used by potential trading partners to prove that they are interacting with the Web services properly. Passing this test could be part of the approval process for new vendors.

Universal Description, Discovery, and Integration (UDDI)

Now that we have described why a registry is a useful thing, we need to address the issue of what the registry will look like. It would do us little good if everyone used a different format for their registry. We would just move the interoperability problem from our internal systems to the registries.

Fortunately, a consensus has emerged on how to control the process of evolving a directory standard. Whenever standardization is needed, there are always three options:

- **Standardize on one vendor's offering**—Vendors are always trying to get this approach to work, but only with their products. The problem with this is that it shuts all other vendors out, causing them to become hostile. In addition, it removes the incentive for improvement if the winning vendor becomes complacent.
- **Wait for a standards body to devise a solution**—The most relevant word is "wait." Standards bodies are notoriously slow moving. Academic types who are more interested in getting it right than in getting it done often dominate these committees.
- **Establish a vendor-sponsored standard**—The computing world is always suspicious of vendor-sponsored standards because they are often thinly veiled attempts to get the world to standardize on their offering. These bodies move quickly, though, because they are often run by industry leaders who are under orders not to dawdle.

NEW TERM A vendor-sponsored registry standard called the *Universal Description, Discovery, and Integration (UDDI)* has emerged and taken a dominant role in this area. UDDI was the brain child of Ariba, IBM, Intel, Microsoft, and SAP. In 2002, the OASIS standards group took over UDDI from UDDI.org. BEA, Cincom, CA, E2Open, Entrust, Fujitsu, HP, IBM, Intel, IONA, Microsoft, Novell, Oracle, SAP, Sun Microsystems, and hundreds of other companies have endorsed it. The UDDI co-chairs are from IBM and Microsoft. The most notable company to sign on is Sun Microsystems. It seems that whenever Sun and Microsoft agree on a standard, the development world follows along. The reasoning seems to be that anytime those two

companies agree on anything, given their rancorous past, it must be good. Additionally, no fees or licenses are required to use this technology.

UDDI is described as follows:

- It is a standardized, transparent mechanism for describing services.
- It describes simple methods for invoking the service.
- It specifies an accessible central registry of services.

An additional feature is that the registries are based on a confederated model whereby they replicate each other. If you register your Web service with one registry, the other registries will receive your information the next time they synchronize. This means that all publicly published Web services can be found by accessing any of the public registries.

The reason that UDDI is acceptable to all the vendors mentioned previously is that it is built on the same SOAP standards that ordinary Web services are. This means that a registry can be written in and accessed by any computer language running on any hardware platform running any operating system. Every vendor is able to create tools to interact with these registries.

Version 3 of the UDDI standard is now published and is being implemented by the IT community.

The UDDI Architecture

The UDDI itself is configured as a replicated collection of Web services. All the public directories replicate information posted on any of them. This ensures that you can access all the public Web services by accessing only one of them.

The information in a registry is composed of three types of entries: white, yellow, and green pages. The analogy to a telephone listing is not too subtle:

- **White pages**—Contain basic contact information such as contact names and phone numbers, plus Dun & Bradstreet D-U-N-S Number. They also contain human-readable information to assist in evaluating the Web service offering.
- **Yellow pages**—Contain classifications using various taxonomy systems. In UDDI version 1, three built-in taxonomies were provided—NAICS industry categorizations, UNSPC project and service categorizations, and ISO-3166-2 geographic taxonomies. In UDDI version 3, external taxonomy systems can be employed so that niche fields can employ their own special classification codes while maintaining the accuracy of the data by verifying all entries against lists of valid entries. Among these classifications is a geography code for situations in which proximity is important.

- **Green pages**—Contain the details on how to program a client that can invoke the service. You can think of the green pages as a type of data dictionary for the service.

UDDI registries work just like other Web services, as shown in Figure 11.4.

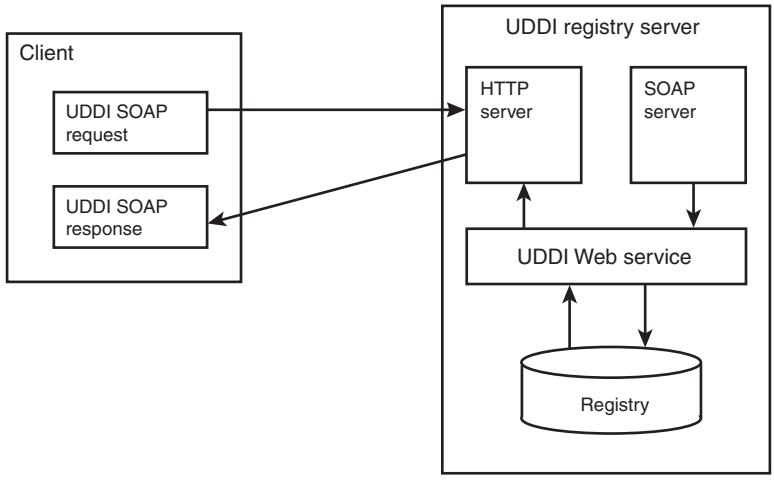

FIGURE 11.4
A UDDI registry operates as a Web service.

All the APIs in the UDDI specification are defined in XML, placed inside SOAP envelopes, and sent over HTTP. In addition, client requests that entail modifying data are required to be secured and authenticated.

One critical part of the whole UDDI search strategy is the technical model (tModel). If you know the Web service that you want, its URL, and that it accepts HTTP, you could, in theory, connect to it. In some instances, however, the Web service is expecting to receive an XML document in a very specific format. The tModel provides metadata about additional specifications, its name, publisher, and the URL of its schema. The tModel commits the Web service to properly process all messages that are sent in this format.

The tModel is not expected to be unique for each Web service. If two Web services are capable of processing the same messages, they would have the same tModel. This could reduce the amount of programming needed to access Web services by allowing the code written for one tModel to be used to access another.

Listing 11.1 shows a summary of the UDDI version 1 Inquiry API.

LISTING 11.1 The UDDI Inquiry Methods

```
Find_business()
Find_service()
Find_binding()
Find_tModel()
Get_businessDetail()
Get_serviceDetail()
Get_bindingDetail()
Get_tModelDetail()
Get_registeredInfo()
```

A programmer would implement these method calls to gather information from the registry about candidate Web services. Listing 11.2 shows the publishing operations in the UDDI API.

LISTING 11.2 The UDDI Publishing Methods

```
save_business()
save_service()
save_binding()
save_tModel()

delete_business()
delete_service()
delete_binding()
delete_tModel()
get_registeredInfo()
get_authToken()
discard_authToken()
```


You make calls to the registry using SOAP just as you would with any other Web service. You use it by downloading a WSDL and contacting the registry as a Web service.

The `authToken` provides an additional level of security. This token is given to a user who has identified himself to the registry's satisfaction to be the legal owner of an organization's registry information.

Programming with UDDI

Programming with the UDDI API is really the same as programming with any Web service. A client must encode messages in SOAP format and send them. When the response arrives, the client must decode the message and extract information from it.

This doesn't mean, however, that you will soon see large numbers of SOAP programmers walking around with SOAP manuals under their arms or college courses teaching

SOAP. In fact, most programmers will use their own favorite programming languages. In the Java world, UDDI4J is a popular library for accessing UDDI registries. Listing 11.3 shows a fragment of the UDDI schema that describes the `businessEntity`.

LISTING 11.3 The `businessEntity` Schema

```
<element name="businessEntity">
  <type content="elementOnly">
   <group order="seq">
     <element ref="discoveryURLs" minOccurs="0" maxOccurs="1"/>
     <element ref="name">
     <element ref="description" minOccurs="0" maxOccurs="1"/>
     <element ref="contacts" minOccurs="0" maxOccurs="1"/>
     <element ref="businessServices" minOccurs="0" maxOccurs="1"/>
     <element ref="identifierBag" minOccurs="0" maxOccurs="1"/>
     <element ref="categoryBag" minOccurs="0" maxOccurs="1"/>
   </group>
   <attribute name="businessKey" minOccurs="1" type="string"/>
   <attribute name="operator" type="string"/>
   <attribute name="authorizedName" type="string"/>
  </type>
</element>
```

The information sent in the SOAP message will be sent in accordance with this format. To make it easier to work with, UDDI4J has defined a class called `com.ibm.uddi.datatype.business.BusinessEntity`. Part of that class is shown in Listing 11.4.

LISTING 11.4 The `BusinessEntity` Class

```
public class BusinessEntity extends UDDIElement
{
  public static final String UDDI_TAG = "businessEntity";
  protected Element base = null;
  String businessKey = null;
  String operator = null;
  String authorizedName = null;
  DiscoveryURLS discoveryURLs = null;
  Name name = null;
  Contacts contacts = null;
  BusinessServices businessServices = null;
  IdentifierBag identifierBag = null;
  CategoryBag = null;
  // Vector of Description objects
  Vector description = new Vector();
  ...
}
```

You can see how a Java programmer would have an easier time populating the class in Listing 11.4 than he would generating the text, as shown in Listing 11.3. The UDDI4J provides the translation from the familiar Java class structure to the less-familiar SOAP messages.

Types of Discovery

NEW TERM There are two visions of how discovery can take place using a UDDI registry. The first, *design-time discovery*, involves quite a bit of human intervention and some programming. The second, *runtime discovery*, is much more automatic and involves quite a bit of tricky programming.

Design-time Discovery

If a company is considering using a Web service, it needs to decide which one. One way to do this is by a manual discovery process. The basic steps of this type of discovery are shown here:

1. The business analysts browse the UDDI to find candidate Web services.
2. A programmer searches the UDDI registry for a `bindingTemplate` that he wants to connect with.
3. The programmer writes a client program based on the `bindingTemplate` and the tModel that it contains. The tModel provides information about complex data (XML documents) that must be transferred to use the service.
4. The program accesses the Web service, transfers the document, and then gets the response.
5. The programmer places the client in production.

Runtime Discovery

In the future, the manual steps will be eliminated and runtime discovery will be more common. This more advanced, and futuristic, solution involves the discovery and interconnection of clients to Web services. The steps needed to perform this type of search are listed here:

1. Business analysts use the GUI to specify what kind of Web services they want.
2. A discovery program searches the UDDI registry for a `bindingTemplate` that the client can connect with, given the protocols that it already supports.
3. The discovery program searches the `bindingTemplate` for the tModel that it contains. If the client program can send requests to this tModel, it is called by the discovery program.

4. The client program accesses the Web service, transfers the document, and then gets the response.

The biggest problem in this scenario is trust. In a world of hackers, it is hard to see how our businesses could trust any Web service enough to allow hands-free interconnection. Keep in mind, though, that trust can be earned by applying the proper safeguards. In the 1920s, no one could have imagined that workers would one day accept a piece of paper as wages instead of an envelope of money. A doctor's board certification gives us enough confidence to allow him to operate on our child less than an hour after meeting him.

The building up of trust will take some time, but slowly, it will progress to the point at which we are able to trust more and more.

In spite of all the challenges that face this vision, UDDI has a bright future. As Brent Sleeper of the Stencil Group said, "UDDI will succeed because its technical underpinnings work for the geeks…"

Summary

In this hour, we covered the need for advertising the capabilities of your Web services. Following that, you learned why a registry is a good way to accomplish this.

Next, you learned about the Universal Description, Discovery, and Integration (UDDI) standard. We covered the origins of this standard and its current level of acceptance. Finally, we stepped through some scenarios that showed how a UDDI registry could be used to discover Web services both manually and automatically.

Q&A

Q Hasn't the whole idea of automatic discovery been rejected by the business community?

A No, it hasn't been greeted with the same enthusiasm as SOAP and WSDL, but it hasn't been rejected either. Everyone seems to put the UDDI part of Web services on the future to-do list instead on today's list.

Q Isn't the whole notion of "promiscuous e-commerce" a hacker's paradise?

A Yes. The safeguards are not really in place to allow the transfer of millions of dollars to a company found on a UDDI registry automatically. This doesn't mean that it will never be safe, however.

Q Is a private registry really useful?

A Yes, but only for companies that are large and diversified, such as GE or Lockheed-Martin. There are probably simpler ways of discovering each other's systems in smaller organizations.

Workshop

This section is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1. What is a registry?
2. Why does an internal Web service need to be advertised?
3. What does UDDI stand for?
4. What is holding back the idea of automatic discovery and interconnection of Web services?

Quiz Answers

1. A registry is a kind of special-purpose database that contains information about Web services.
2. In large organizations, major systems can exist, but not every one knows about them. A private UDDI registry could keep track of all the Web services in an organization.
3. It stands for Universal Description, Discovery, and Integration.
4. Security and the fear of hackers make today's businesses prefer to have a human involved in the process of discovery and authorization.

Activities

1. List the three different types of registries and describe the different motivation behind the creation of each of them.
2. Go to Microsoft's test registry at `http://uddi.microsoft.com/` to experiment and gain experience with UDDI registries.