UDDI Fundamentals

- Discovering Services Dynamically
- UDDI API
- UDDI Information Model

What is UDDI?

- The Universal Description, Discovery and Integration interface provides a **registry**, an **API** and an **information model** for discovering services dynamically
- UDDI is administered by the OASIS group

You can find more information about UDDI at www.uddi.org.

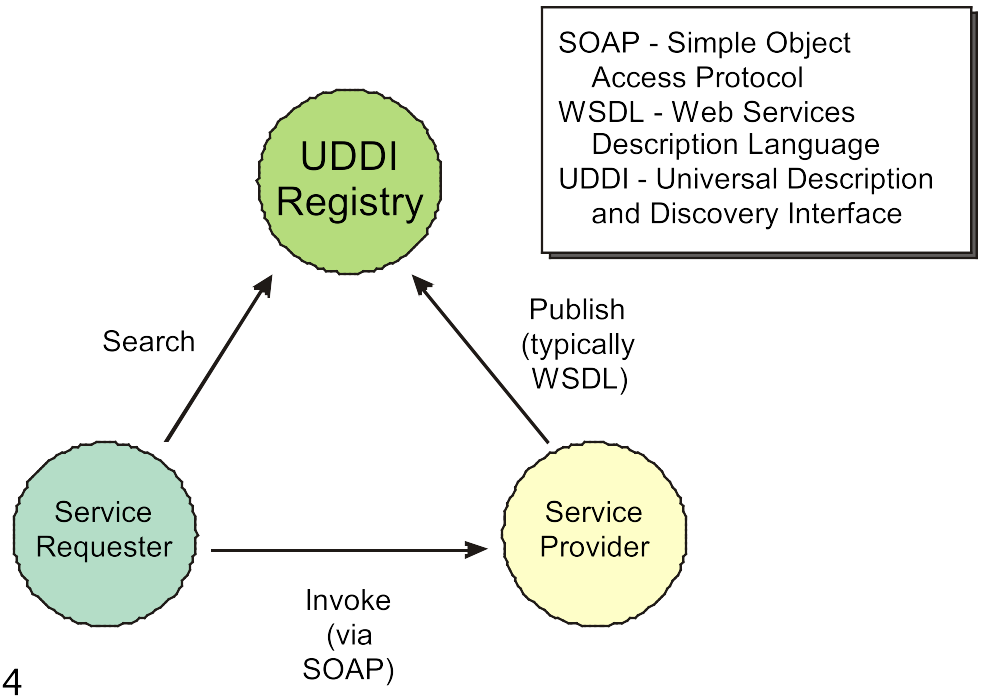
UDDI History

- UDDI evolved from collaborations of IBM, Ariba and Microsoft
- Version 1.0 was released in September 2000
- Version 2.0 was released in June 2001
- Version 3.0 was release in December 2001

Many current UDDI implementations still are at version 2.0.

UDDI and SOA

- UDDI is an important part of a service oriented architecture system that uses Web services

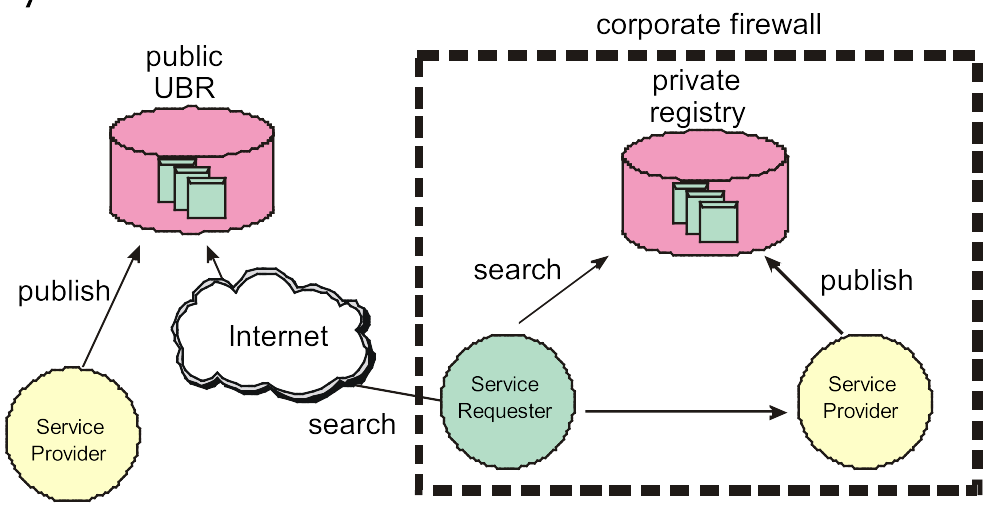


2 - 4

An important facet of SOA and of Web services is dynamic discovery of services at runtime. UDDI is the standard technology for such discovery in the Web services world.

Public and Private Registries

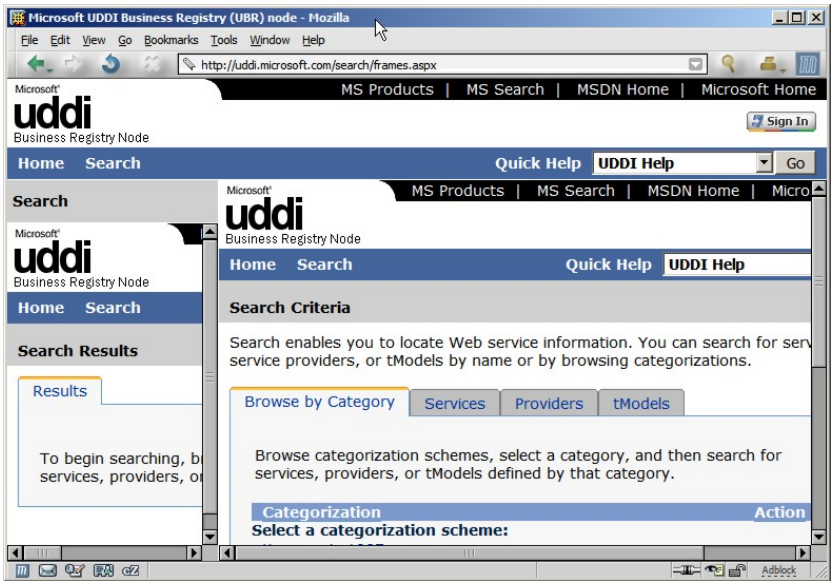
- An enterprise or set of business partners can run a **private registry** to allow for dynamic discovery of services within the enterprise
- The public **Universal Business Registry** is implemented by several companies for use by anyone



The intent of UBR is primarily to allow businesses to advertise themselves and the services they provide –– since it's not moderated, it's not very reliable, however.

Instead, most enterprises will most likely run private UDDI registries within their firewall and perhaps allow trusted business partners access as well. Most J2EE application server vendors, including IBM WebSphere and BEA WebLogic provide UDDI servers as part of their containers.

The Universal Business Registry



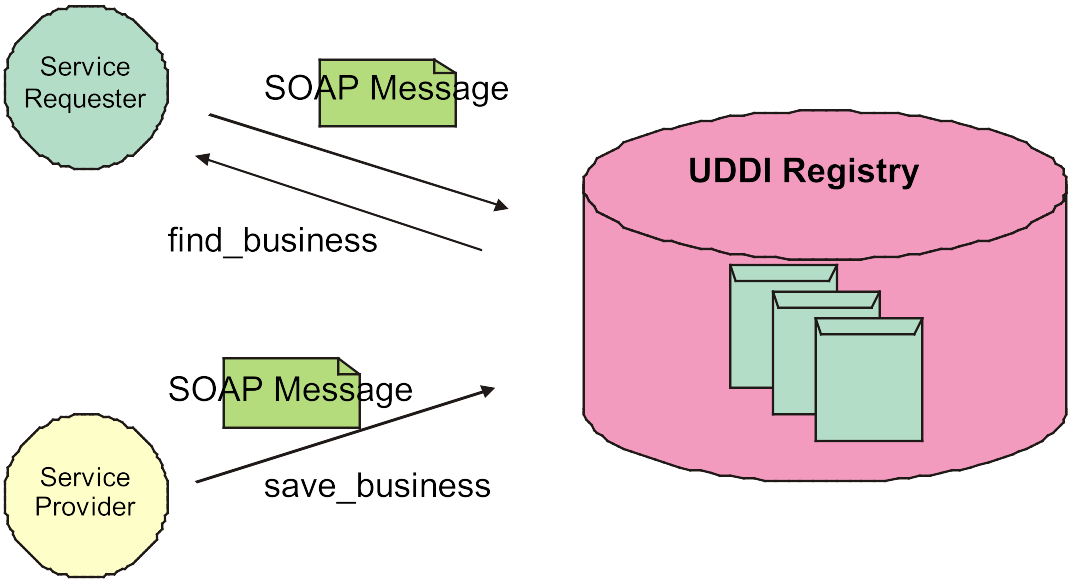
Note that as of January 2006, the UBR is defunct.

The UBR is operated by several companies, including IBM, Microsoft, SAP and NTT. Here we show a screenshot of the Microsoft Web interface for the UBR.

UDDI defines a replication protocol that is implemented by the UBR. This means that if you register a business or a service using say Microsoft's Web interface, the information you enter will be replicated to the other servers so you could use another company's interface, say IBM's, to query the information.

Invoking UDDI

- UDDI itself is a distributed service that uses SOAP messages



2 - 7

The UDDI API is a textbook example of where SOAP is useful -- the API itself is platform, language and vendor neutral.

However, as we will see later, in most cases you will not need to send raw SOAP messages to work with UDDI since there are several higher-level APIs available.

UDDI Taxonomies

- UDDI recognizes several categorization schemes or **taxonomies** that allow requesters to search by category
  - NAICS
  - UNSPEC
  - ISO 3166 Geographic

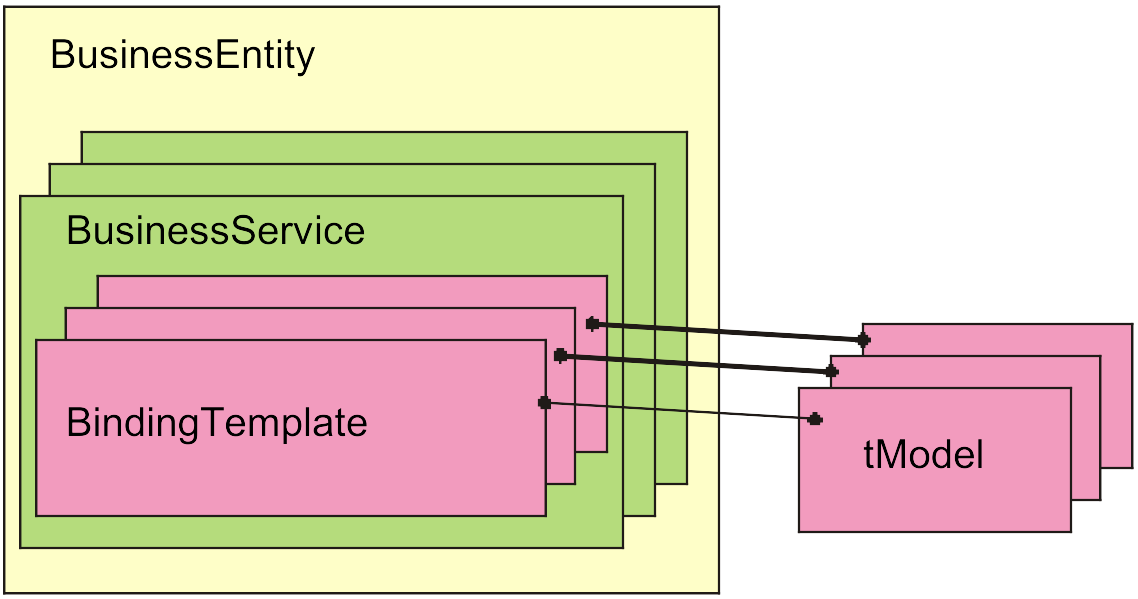
Sample NAICS 1997 Codes

```
. . .
51      Information
511      Publishing Industries
5111      Newspaper, Periodical, Book, and Database Publishers
51111      Newspaper Publishers
51112      Periodical Publishers
51113      Book Publishers
51114      Database and Directory Publishers
51119      Other Publishers
511191      Greeting Card Publishers
511199      All Other Publishers
5112      Software Publishers
51121      Software Publishers
. . .
```

You can attach categories to almost everything in UDDI, including business entries, service entries and so forth. In most cases, the more category information you provide, the more likely it is that requestors will find your data.

NAICS is an acronym for North American Industry Classification System.

UDDI Information Model



The UDDI registry is organized into "objects", the structure of which is show here. The "BusinessEntity" represents an enterprise and contains the business name, contacts, categories and so forth.

A BusinessEntity can contain zero or more "BusinessService" objects, each of which provides high-level information about a service, including the service's name and categories.

Associated with a BusinessService, is zero or more "BindingTemplate" and "tModel" pairs, which provide technical information about the service, including the service's "access point", which is the URL to which it responds.

The BusinessEntity

- The **BusinessEntity** entry describes the publisher of the service

The first important field to discuss is the "BusinessKey", which is a Universally Unique Identifier (UUID) assigned by the registry when the BusinessEntry is created.

The "CategoryBag" allows you to assign zero or more categories to the business, typically using one of the taxonomies discussed earlier.

The "IdentifierBag" lets you assign zero or more unique identifiers for the business, for example a DUNS number.

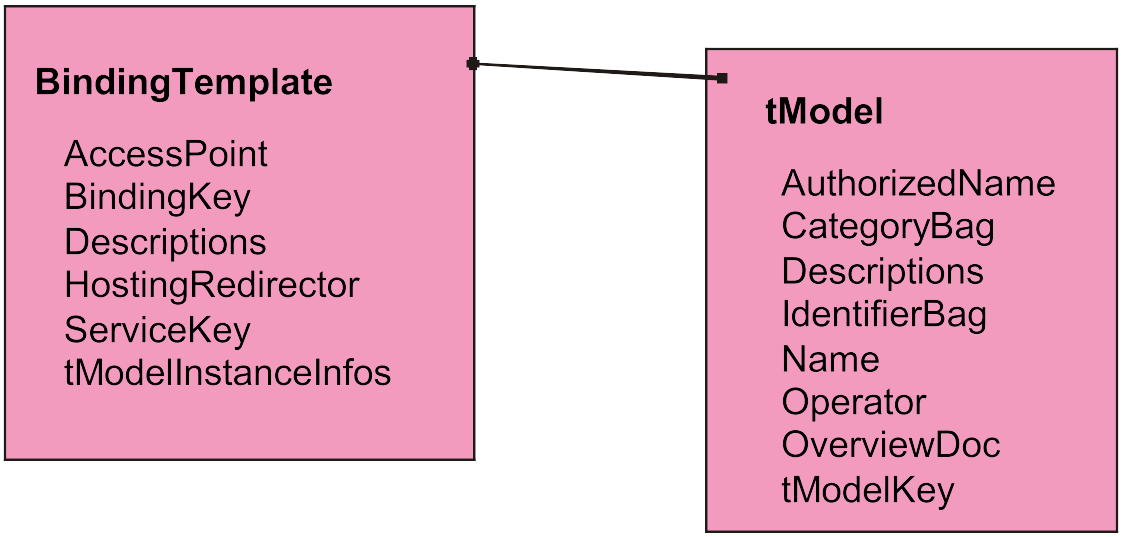
The BusinessService

- The **BusinessService** entry describes high-level information about a service

The important field here is the "ServiceKey", which is a UUID assigned by the registry when the service is published to the registry. This is useful since it's possible to search uniquely for a service, given its key.

The BindingTemplate and tModel

- The **BindingTemplate** and **tModel** provide technical information about the service

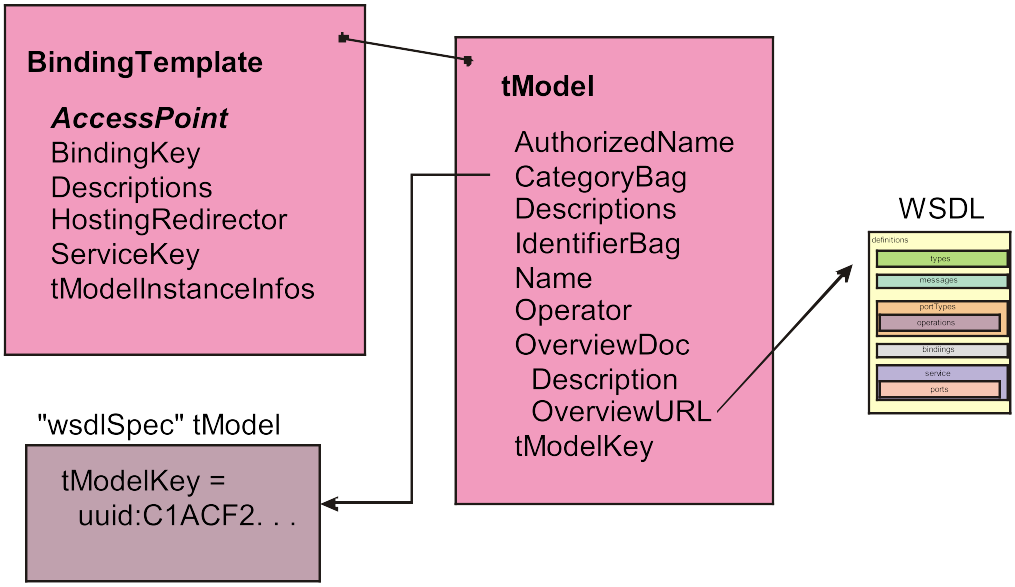


The useful fields here are the "AccessPoint" in the BindingTemplate, which contains the endpoint address of the service, and the "OverviewDoc" in the tModel, in which you can store the URL of the WSDL for the service.

Note that both structures contain UUIDs that uniquely identify them –– again, that lets requesters perform unique searches if they know the UUID.

Integrating UDDI and WSDL

- Since UDDI predates WSDL, storing a reference in WSDL is not as straightforward as one might think



Since UDDI was not explicitly defined for use by Web services, we have to "kludge" a bit to make it work. The trick is to store a reference to the WSDL in the "OverviewURL" field which is a sub-field of "OverviewDoc" in the tModel. Then, by convention, to indicate that we've stored a WSDL reference there, we attached a well-known, predefined "wsdlSpec" tModel as a category on the tModel in question.

Note that this is how it works in UDDI version 2 -- in version 3, there's a more elegant way of indicating that a tModel contains a WSDL reference.

Integrating UDDI and WSDL, cont'd

- OASIS has published a set of best practices for publishing WSDL in UDDI:

tModel	
name	WSDL's target namespace
overviewURL	URL of WSDL's binding element
categoryBag	Reference standard "wsdlSpec" tModel

binding template	
accessPoint	soap:location value from WSDL
tModelInstanceInfo	Reference the tModel

For more information, see:

http://www.uddi.org/bestpractices.html

http://www-128.ibm.com/developerworks/webservices/library/ws-wsdl/

UDDI Publishing API

add_publisherAssertions	Define relationships between business entities
delete_binding	Remove a binding template
delete_business	Remove a business entity
delete_publisherAssertions	Un-define a relationship between business entities
delete_service	Remove s business service
delete_tModel	Remove a tModel
discard_authToken	Get rid of an previously obtained authorization token
get_assertionStatusReport	Retrieve info about assertions
get_authToken	Login to the server and retrieve an authorization token
get_publisherAssertions	Retrieve list of assertions
get_registeredInfo	Retrieve a short list of tModels associated with an entity
save_binding	Create or update a binding template
save_business	Create or update a business entity
save_service	Create or update a business service
save_tModel	Create or update a tModel
set_publisherAssertions	Assign assertions

In most cases, before you can publish information to a registry, you must authenticate with the registry. If the registry provides no native login capability, you can use the "get_authToken" message to login and retrieve a token that you can pass on subsequent messages.

Note that the "save_xxxx" messages will either create a new entry or update an existing one –– these messages accept the unique identifier for the structure in question (e.g. a tModelKey). If you don't provide the unique ID, the registry will create a new entry and return the unique ID.

Sample UDDI Publishing Message

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope
   xmlns="http://schemas.xmlsoap.org/soap/envelope">
  <Body>
    <save_business generic="2.0"
         xmlns="urn:uddi-org:api_v2" >
     <authInfo>xxxyyyzzzzz</authInfo>
     <businessEntity businessKey="">
       <name>Goliath, Inc.</name>
     </businessEntity>
   </save_business>
  </Body>
</Envelope>
```


Here we show an example publishing message –– save_business. Note that since we didn't provide a businessKey, the registry assumes that we are publishing a new business entity, and will return a new key in the response message.

Note also that we are assuming here that we've previously authenticated with the registry and we are providing the "authInfo" token returned by a pervious call to "get_authToken".

UDDI Inquiry API

find_binding	Search for binding templates
find_business	Search for business entities
find_relatedBusinesses	Search for associated businesses
find_service	Search for business services
find_tModel	Search for tModels
get_bindingDetail	Given a binding template key, get detailed info
get_businessDetail	Given a business entity key, get detailed info
get_businessDetailExt	Same as above, but for external registries
get_serviceDetail	Given a business service key, get detailed info
get_tModelDetail	Given a tModel key, get detailed info

Unlike the publishing API, the inquiry API normally doesn't require authentication. Note that in many cases, it takes two messages to get detailed information from the registry.

Sample UDDI Inquiry Message

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope
   xmlns="http://schemas.xmlsoap.org/soap/envelope">
  <Body>
    <find_business generic="2.0"
          xmlns="urn:uddi-org:api_v2">
      <name>Goliath, Inc</name>
    </find_business>
  </Body>
</Envelope>
```


Here we show a sample inquiry –– "find_business". The response message will contain the list of business entities that match the specified name. Note that you can use the wild–card character "%" to match more than one business name.

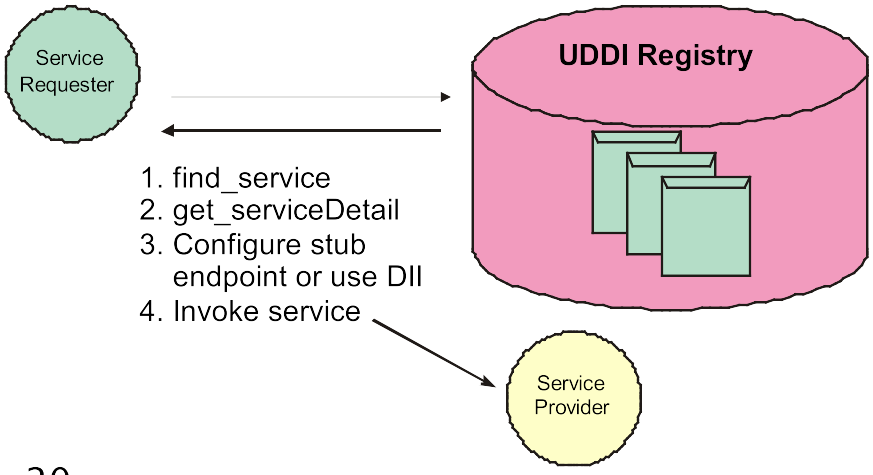
Accessing UDDI From Java

- There are several Java APIs for invoking UDDI:
  - IBM UDDI4J
  - Sun JAX-R
  - WebLogic UDDI Client API
- JAX-R is a required API in J2EE 1.4

The good news is that there are several choices -- the bad news is that as of this writing, none of the APIs has established itself as the de facto standard.

Writing Dynamic Service Requesters

- A truly dynamic client can use UDDI to find a service and then use DII to invoke the service
- Another option would be to use static stubs or dynamic proxies and query the service endpoint from the UDDI registry



This is the real benefit of a service-oriented architecture -- we can write clients that are insensitive to changes in the service provider. In other words, instead of having the client hard-code information about the service, the client can determine the information at runtime -- that way if something about the service changes, the client need not be re-written.

Chapter Summary

In this chapter, you learned:

- How UDDI lets us build systems that discover information at runtime
- About the UDDI API
- About the UDDI information model