UDDI – UNIVERSAL DESCRIPTION, DISCOVERY INTERFACE

UDDI is an **XML-based standard** for describing, publishing, and finding web services.

UDDI stands for **Universal Description, Discovery, and Integration.**

UDDI is a **specification** for a distributed registry of web services.

UDDI is a **platform-independent, open framework.**

UDDI can communicate via SOAP, CORBA, Java RMI Protocol.

UDDI uses Web Service Definition **Language(WSDL) to describe interfaces** to web services.

UDDI is seen with SOAP and WSDL as one of the three **foundation standards** of web services.

UDDI is an open industry initiative, enabling businesses to discover each other and define how they interact over the Internet.

UDDI has two sections –

A **registry** of all web service's metadata, including a pointer to the WSDL description of a service.

A set of **WSDL port type definitions** for manipulating and searching that registry.

UDDI ELEMENTS

A business or a company can register three types of information into a UDDI registry. This information is contained in three elements of UDDI.

These three elements are −

White Pages,

Yellow Pages, and

Green Pages.

White Pages

White pages contain –

Basic information about the company and its business.

Basic contact information including business name, address, contact phone number, etc.

A Unique identifiers for the company tax IDs. This information allows others to discover your web service based upon your business identification.

Yellow Pages

Yellow pages contain more details about the company. They include descriptions of the kind of electronic capabilities the company can offer to anyone who wants to do business with it.

Yellow pages uses commonly accepted industrial categorization schemes, industry codes, product codes, business identification codes and the like to make it easier for companies to search through the listings and find exactly what they want.

Green Pages

Green pages contains technical information about a web service. A green page allows someone to bind to a Web service after it's been found.

It includes −

The various interfaces

The URL locations

Discovery information and similar data required to find and run the Web service.

The UDDI technical architecture consists of three parts −

UDDI Data Model

UDDI Data Model is an XML Schema for describing businesses and web services.

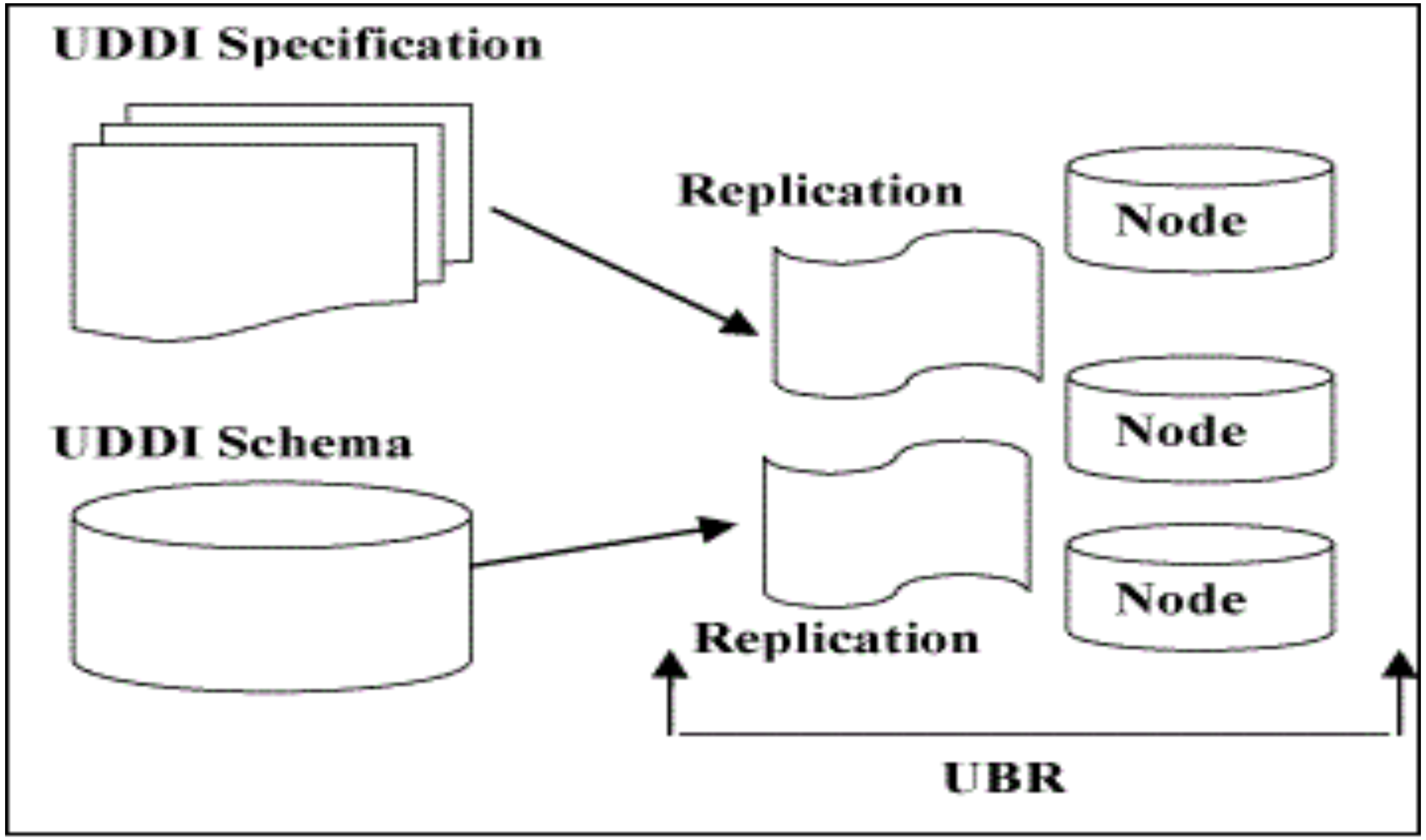
UDDI API Specification

It is a specification of API for searching and publishing UDDI data.

UDDI Cloud Services

These are operator sites that provide implementations of the UDDI specification and synchronize all data on a scheduled basis.

The UDDI technical architecture



The UDDI Business Registry (UBR), also known as the Public Cloud, is a conceptually single system built from multiple nodes having their data synchronized through replication.

The current cloud services provide a logically centralized, but physically distributed, directory. It means the data submitted to one root node will automatically be replicated across all the other root nodes. Currently, data replication occurs every 24 hours.

UDDI cloud services are currently provided by Microsoft and IBM. Ariba had originally planned to offer an operator as well, but has since backed away from the commitment. Additional operators from other companies, including Hewlett-Packard, are planned for the near future.

It is also possible to set up private UDDI registries. For example, a large company may set up its own private UDDI registry for registering all internal web services. As these registries are not automatically synchronized with the root UDDI nodes, they are not considered as a part of the UDDI cloud.

UDDI includes an XML Schema that describes the following data structures −

businessEntity

businessService

bindingTemplate

tModel

publisherAssertion

UDDI includes an XML Schema that describes the following data structures −

businessEntity

businessService

bindingTemplate

tModel

publisherAssertion

Here is an example of a fictitious business's UDDI registry entry −

```
<businessEntity businessKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40"
   operator = "http://www.ibm.com" authorizedName = "John Doe">
   <name>Acme Company</name>
   <description>
      We create cool Web services
   </description>

   <contacts>
      <contact useType = "general info">
         <description>General Information</description>
         <personName>John Doe</personName>
         <phone>(123) 123-1234</phone>
         <email>jdoe@acme.com</email>
      </contact>
   </contacts>
```

```
<businessServices>
      ...
   </businessServices>
   <identifierBag>
      <keyedReference tModelKey = "UUID:8609C81E-EE1F-4D5A-B202-3EB13AD01823"
         name = "D-U-N-S" value = "123456789" />
   </identifierBag>

   <categoryBag>
      <keyedReference tModelKey = "UUID:C0B9FE13-179F-413D-8A5 5004DB8E5BB2"
         name = "NAICS" value = "111336" />
   </categoryBag>
</businessEntity>
```

businessService Data Structure

The business service structure represents an individual web service provided by the business entity. Its description includes information on how to bind to the web service, what type of web service it is, and what taxonomical categories it belongs to.

Here is an example of a business service structure for the Hello World web service.

```
<businessService serviceKey = "uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
   businessKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">
   <name>Hello World Web Service</name>
   <description>A friendly Web service</description>
   <bindingTemplates>
      ...
   </bindingTemplates>
   <categoryBag />
</businessService>
```

Notice the use of the Universally Unique Identifiers (UUIDs) in the *businessKey* and *serviceKey* attributes. Every business entity and business service is uniquely identified in all the UDDI registries through the UUID assigned by the registry when the information is first entered.

<u>**bindingTemplate Data Structure**</u>

Binding templates are the technical descriptions of the web services represented by the business service structure. A single business service may have multiple binding templates. The binding template represents the actual implementation of the web service.

Here is an example of a binding template for Hello World.

```
<bindingTemplate serviceKey = "uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
  bindingKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">
  <description>Hello World SOAP Binding</description>
  <accessPoint URLType = "http">http://localhost:8080</accessPoint>

  <tModelInstanceDetails>
    <tModelInstanceInfo tModelKey = "uuid:EB1B645F-CF2F-491f-811A 4868705F5904">
      <instanceDetails>
        <overviewDoc>
          <description>
            references the description of the WSDL service definition
          </description>
```

```
 <overviewURL>
                  http://localhost/helloworld.wsdl
               </overviewURL>
            </overviewDoc>
         </instanceDetails>
      </tModelInstanceInfo>
   </tModelInstanceDetails>
</bindingTemplate>
```

As a business service may have multiple binding templates, the service may specify different implementations of the same service, each bound to a different set of protocols or a different network address.

tModel Data Structure

tModel is the last core data type, but potentially the most difficult to grasp. tModel stands for technical model.

tModel is a way of describing the various business, service, and template structures stored within the UDDI registry. Any abstract concept can be registered within the UDDI as a tModel. For instance, if you define a new WSDL port type, you can define a tModel that represents that port type within the UDDI. Then, you can specify that a given business service implements that port type by associating the tModel with one of that business service's binding templates.

Here is an example of a tModel representing the Hello World Interface port type.

```
<tModel tModelKey = "uuid:xyz987..." operator = "http://www.ibm.com"
   authorizedName = "John Doe">
   <name>HelloWorldInterface Port Type</name>
   <description>
      An interface for a friendly Web service
   </description>

   <overviewDoc>
      <overviewURL>
         http://localhost/helloworld.wsdl
      </overviewURL>
   </overviewDoc>
</tModel>
```

publisherAssertion Data Structure

This is a relationship structure putting into association two or more businessEntity structures according to a specific type of relationship, such as subsidiary or department.

The publisherAssertion structure consists of the three elements: fromKey (the first businessKey), toKey (the second businessKey), and keyedReference.

The keyedReference designates the asserted relationship type in terms of a keyName keyValue pair within a tModel, uniquely referenced by a tModelKey.

```
<element name = "publisherAssertion" type = "uddi:publisherAssertion" />
<complexType name = "publisherAssertion">
  <sequence>
    <element ref = "uddi:fromKey" />
    <element ref = "uddi:toKey" />
    <element ref = "uddi:keyedReference" />
  </sequence>
</complexType>
```

A registry is of no use without some way to access it. The UDDI standard version 2.0 specifies two interfaces for service consumers and service providers to interact with the registry.

Service consumers use Inquiry Interface to find a service, and service providers use Publisher Interface to list a service.

The core of the UDDI interface is the UDDI XML Schema definitions. These define the fundamental UDDI data types through which all the information flows.

The Publisher Interface

The Publisher Interface defines sixteen operations for a service provider managing its entries in the UDDI registry −

get_authToken − Retrieves an authorization token. All of the Publisher interface operations require that a valid authorization token be submitted with the request.

discard_authToken − Tells the UDDI registry to no longer accept a given authorization token. This step is equivalent to logging out of the system.

save_business − Creates or updates a business entity's information contained in the UDDI registry.

save_service − Creates or updates information about the web services that a business entity provides.

save_binding − Creates or updates the technical information about a web service's implementation.

save_tModel – Creates or updates the registration of abstract concepts managed by the UDDI registry.

delete_business – Removes the given business entities from the UDDI registry completely.

delete_service – Removes the given web services from the UDDI registry completely.

delete_binding – Removes the given web services technical details from the UDDI registry.

delete_tModel – Removes the specified tModels from the UDDI registry.

get_registeredInfo – Returns a summary of everything the UDDI registry is currently keeping track of for the user, including all businesses, all services, and all tModels.

set_publisherAssertions – Manages all of the tracked relationship assertions associated with an individual publisher account.

add_publisherAssertions – Causes one or more publisherAssertions to be added to an individual publisher's assertion collection.

delete_publisherAssertions – Causes one or more publisherAssertion elements to be removed from a publisher's assertion collection.

get_assertionStatusReport – Provides administrative support for determining the status of current and outstanding publisher assertions that involve any of the business registrations managed by the individual publisher account.

get_publisherAssertions – Obtains the full set of publisher assertions that is associated with an individual publisher account.

The Inquiry Interface

The inquiry interface defines ten operations for searching the UDDI registry and retrieving details about specific registrations −

find_binding − Returns a list of web services that match a particular set of criteria based on the technical binding information.

find_business − Returns a list of business entities that match a particular set of criteria.

find_ltservice − Returns a list of web services that match a particular set of criteria.

find_tModel − Returns a list of tModels that match a particular set of criteria.

get_bindingDetail − Returns the complete registration information for a particular web service binding template.

get_businessDetail − Returns the registration information for a business entity, including all services that entity provides.

get_businessDetailExt − Returns the complete registration information for a business entity.

get_serviceDetail − Returns the complete registration information for a web service.

get_tModelDetail − Returns the complete registration information for a tModel.

find_relatedBusinesses − Discovers businesses that have been related via the uddi-org:relationships model.

UDDI- USAGE EXAMPLE

Consider a company XYZ wants to register its contact information, service description, and online service access information with UDDI. The following steps are necessary −

Choose an operator with which to work. Each operator has different terms and conditions for authorizing access to its replica of the registry.

Build or otherwise obtain a UDDI client, such as those provided by the operators.

Obtain an authentication token from the operator.

Register information about the business. Include as much information as might be helpful to those searching for matches.

Release the authentication token.

Use the inquiry APIs to test the retrieval of the information, including binding template information, to ensure that someone who obtains it can use it successfully to interact with your service.

Fill in the tModel information in case someone wants to search for a given service and find your business as one of the service providers.

Update the information as necessary to reflect the changing business contact information and new service details, obtaining and releasing a new authentication token from the operator each time. Whenever you need to update or modify the data you've registered, you have to go back to the operator with which you have entered the data.

The following examples will show how the XYZ Company would register its information

Creating Registry

After obtaining an authentication token from one of the operators Microsoft, for example the XYZ.com developers decide what information to publish to the registry and use one of the UDDI tools provided by Microsoft. If necessary, the developers can also write a Java, C#, or VB.NET program to generate the appropriate SOAP messages. Here is an example.

```
POST /save_business HTTP/1.1
Host: www.XYZ.com
Content-Type: text/xml; charset = "utf-8"
Content-Length: nnnn
SOAPAction: "save_business"

<?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">
      <businessKey = "">
      </businessKey>
```

```
<name>
        XYZ, Pvt Ltd.
      </name>

      <description>
        Company is involved in giving Stat-of-the-art....
      </description>

      <identifierBag> ... </identifierBag>
      ...
    </save_business>
  </Body>
</Envelope>
```

This example illustrates a SOAP message requesting to register a UDDI business entity for XYZ Company. The key element is blank, because the operator automatically generates the UUID key for the data structure. Most fields are omitted for the sake of showing a simple example.

Company XYZ can always execute another save_business operation to add to the basic information required to create a business entity.

Retrieving Information

After XYZ Company has updated its UDDI entry with the relevant information, companies that want to become XYZ distributors can look up contact information in the UDDI registry and obtain the service descriptions and the access points for the two Web services that XYZ.com publishes for online order entry: preseason bulk orders and in-season restocking orders.

This example illustrates a sample SOAP request to obtain business detail information about the XYZ Company. Once you know the UUID, or key, for the specific business that's been registered, you can use it in the get_businessDetail API to return specific information about that business.

```
POST /get_businessDetail HTTP/1.1
Host: www.XYZ.com
Content-Type: text/xml; charset = "utf-8"
Content-Length: nnnn
SOAPAction: "get_businessDetail"

<?xml version = "1.0" encoding = "UTF-8" ?>
<Envelope xmlns = "http://schemas/xmlsoap.org/soap/envelope/">
   <Body>
      <get_businessDetail generic = "2.0" xmlns = "urn:uddi-org:api_v2">
         <businessKey = "C90D731D-772HSH-4130-9DE3-5303371170C2">
         </businessKey>
      </get_businessDetail>
   </Body>
</Envelope>
```

The UDDI data model defines a generic structure for storing information about a business and the web services it publishes. The UDDI data model is completely extensible, including several repeating sequence structures of information.

However, WSDL is used to describe the interface of a web service. WSDL is fairly straightforward to use with UDDI.

WSDL is represented in UDDI using a combination of businessService, bindingTemplate, and tModel information.

As with any service registered in UDDI, generic information about the service is stored in the businessService data structure, and information specific to how and where the service is accessed is stored in one or more associated bindingTemplate structures. Each bindingTemplate structure includes an element that contains the network address of the service and has associated with it one or more tModel structures that describe and uniquely identify the service.

When UDDI is used to store WSDL information, or pointers to WSDL files, the tModel should be referred to by convention as type wsdlSpec, meaning that the overviewDoc element is clearly identified as pointing to a WSDL service interface definition.

For UDDI, WSDL contents are split into two major elements the interface file and the implementation file.

The Hertz reservation system web service provides a concrete example of how UDDI and WSDL works together. Here is the <tModel> for this web service −

```
<tModel authorizedName = "..." operator = "..." tModelKey = "...">
  <name>HertzReserveService</name>
  <description xml:lang = "en">
    WSDL description of the Hertz reservation service interface
  </description>

  <overviewDoc>
    <description xml:lang = "en">
      WSDL source document.
    </description>
    <overviewURL>
      http://mach3.ebphost.net/wsdl/hertz_reserve.wsdl
    </overviewURL>
  </overviewDoc>
```

```
<categoryBag>
    <keyedReference tModelKey =
"uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
      keyName = "uddi-org:types" keyValue = "wsdlSpec"/>
  </categoryBag>
</tModel>
```

The key points are −

The overviewURL element gives the URL to where the service interface definition WSDL file can be found. This allows humans and UDDI/WSDL aware tools to locate the service interface definition.

The purpose of the keyedReference element in the categoryBag is to make sure that this tModel is categorized as a WSDL specification document.