3 Web Services Description Language (WSDL)

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
http://www.iks.inf.ethz.ch/
WSDL as an IDL

- WSDL can be best understood when we approach it as an XML version of an IDL that also covers the aspects related to integration through the Internet and the added complexity of Web services.

- An IDL in conventional middleware and enterprise application integration platforms has several purposes:
  - description of the interfaces of the services provided (e.g., RPC)
  - serve as an intermediate representation for bridging heterogeneity by providing a mapping of the native data types to the intermediate representation associated to the IDL in question
  - serve as the basis for development through an IDL compiler that produces stubs and libraries that can be use to develop the application

- A conventional IDL does not include information such as:
  - location of the service (implicit in the platform and found through static or dynamic binding)
  - different bindings (typically an IDL is bound to a transport protocol)
  - sets of operations (since an interface defines a single access point and there is no such a thing as a sequence of operations involved in the same service)
Standard Layers

TCP/IP

HTTP

XML

SOAP

UDDI

WSDL

Messaging

Discovery

Description

User Interface

HTML

©Gustavo Alonso, D-INFK. ETH Zürich.
## WS Standards and Specifications

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transport</strong></td>
<td>HTTP, IIOP, SMTP, JMS</td>
<td></td>
</tr>
<tr>
<td><strong>Messaging</strong></td>
<td>XML, SOAP</td>
<td>WS-Addressing</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>XML Schema, WSDL</td>
<td>WS-Policy, SSDL</td>
</tr>
<tr>
<td><strong>Discovery</strong></td>
<td>UDDI</td>
<td>WS-MetadataExchange</td>
</tr>
<tr>
<td><strong>Choreography</strong></td>
<td>WSCL, WSCI</td>
<td>WS-Coordination</td>
</tr>
<tr>
<td><strong>Business Processes</strong></td>
<td>WS-BPEL, BPML</td>
<td>WSCDL</td>
</tr>
<tr>
<td><strong>Stateful Resources</strong></td>
<td></td>
<td><strong>WS-Resource Framework</strong></td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td>WS-CAF</td>
<td>WS-Atomic Transactions</td>
</tr>
<tr>
<td><strong>Reliable Messaging</strong></td>
<td>WS-Reliability</td>
<td>WS-ReliableMessaging</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>WS-Security, SAML, XACML</td>
<td>WS-Trust, WS-Privacy, WS-SecureConversation</td>
</tr>
<tr>
<td><strong>Event Notification</strong></td>
<td>WS-Notification</td>
<td>WS-Eventing</td>
</tr>
<tr>
<td><strong>Management</strong></td>
<td>WSDM</td>
<td>WS-Management</td>
</tr>
<tr>
<td><strong>Data Access</strong></td>
<td>OGSA-DAI</td>
<td>SDO</td>
</tr>
</tbody>
</table>

©Gustavo Alonso, D-INFK. ETH Zürich.
Once it is possible to interact with any service provider using the standard SOAP protocol, it is still necessary to:

- Describe the services (WSDL, Web Services Description Language)
- Discover the services (UDDI, Universal Description, Discovery and Integration)
Web Services Description Language (WSDL)
What is WSDL?

- The Web Services Description Language specification is in version 2.0
- WSDL 1.1 discusses how to describe the different parts that comprise a Web service interface
  - the type system used to describe the service data model (XML Schema)
  - the messages involved in the interaction with the service
  - the individual operations composed of 4 possible message exchange patterns
  - the sets of operations that constitute a service
  - the mapping to a transport protocol for the messages
  - the location where the service provider resides
  - groups of locations that can be used to access the same service
- It also includes a specification indicating how to bind WSDL to the SOAP, HTTP (POST/GET) and MIME protocols
Versions in time …

WSDL 1.1

- definitions
  - types
  - message
  - portType
    - operation
      - input
      - output
  - binding
  - service
    - port

WSDL 2.0

- description
  - types
  - interface
    - operation
      - input
      - output
  - binding
  - service
    - endpoint
Elements of WSDL 2.0

WSDL document

- Types (type information for the document, e.g., XML Schema)
  - Message 1
  - Message 2
  - Message 3
  - Message 4
  - Message 5
  - Message 6

- Operation 1
- Operation 2
- Operation 3

Interface (abstract service)

- binding 1
  - endpoint 1
- binding 2
  - endpoint 2
- binding 3
  - endpoint 3
- binding 4
  - endpoint 4

Service (the interface in all its available implementations)
Types in WSDL

- The types in WSDL are used to specify the contents of the messages (normal messages and fault messages) that will be exchanged as part of the interactions with the Web service.
- The type system is typically based on XML Schema (structures and data types) - support is mandatory for all WSDL processors.
- An extensibility element can be used to define a schema other than XML Schema.
Types in WSDL (Example)

```xml
<element name="PO" type="tns:POType"/>
<complexType name="POType">
    <all>
        <element name="id" type="string"/>
        <element name="name" type="string"/>
        <element name="items">
            <complexType>
                <all>
                    <element name="item" type="tns:Item" minOccurs="0" maxOccurs="unbounded"/>
                </all>
            </complexType>
        </element>
    </all>
</complexType>
<complexType name="Item">
    <all>
        <element name="quantity" type="int"/>
        <element name="product" type="string"/>
    </all>
</complexType>
<element name="Invoice" type="tns:InvoiceType"/>
<complexType name="InvoiceType">
    <all>
        <element name="id" type="string"/>
    </all>
</complexType>
```

PURCHASE ORDER TYPE

ITEM TYPE

INVOICE TYPE

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001
Messages and Faults

- Messages have a name that identifies them throughout the XML document. Messages are divided into parts, each of them being a data structure represented in XML. Each part must have a type (basic or complex types, previously declared in the WSDL document).

- A WSDL message element matches the contents of the body of a SOAP message. By looking at the types and looking at the message, it is possible to build a SOAP message that matches the WSDL description (and this can be done automatically since the description is XML based and the types also supported by SOAP).

- A message does not define any form of interaction, it is just a message.

- In WSDL 1.0, the structure of a “message” is explicitly defined, listing all of its parts.

- In WSDL 2.0, a “message reference component” is defined as part of an operation and contains three elements:
  - Message label (indicating the message pattern used for the message)
  - Direction (whether it is an inbound or outbound message)
  - Message element (the actual contents of the message expressed in terms of the types previously defined)

- Faults are a special kind of message used to report errors.

```xml
<message name="PO">
  <part name="po" element="tns:PO"/>
  <part name="invoice" element="tns:Invoice"/>
</message>
```
Operations provide the first level of context for the messages. In WSDL 1.0, there are four types of operations:

- **one-way**: the client sends a message to the server
- **request-response**: the client sends a request, the server replies with a response
- **Solicit-response**: the server sends a message and the client replies
- **Notification**: the server sends a message

In WSDL 2.0, an operation is a set of messages and faults. The sequencing and number of messages in the operation is determined by the message exchange pattern.

The style of an operation distinguishes between RPC-like behavior, document oriented message exchange or (in 2.0) set and get of attributes.

Operations can be annotated with features and properties (e.g., reliability, security, routing).

---

ONE-WAY:

```xml
<wsdl:operation name="Purchase">
  <wsdl:input name="Order" message="PO"/>
</wsdl:operation>
```

REQUEST-RESPONSE:

```xml
<wsdl:operation name="Purchase">
  <wsdl:input name="Order" message="PO"/>
  <wsdl:output name="Confirm" message="Conf"/>
  <wsdl:fault name="Error" message="POError"/>
</wsdl:operation>
```
Port Types (interfaces)

- An interface corresponds to the abstract definition of a Web service (abstract because it does not specify any information about where the service resides or what protocols are used to invoke the Web service).
- The Interface is simply a list of operations that can be used in that Web service.
- Operations are not defined by themselves but only as part of an interface.
- In WSDL 2.0 Port Types have been renamed to Interfaces (which also support inheritance).

```xml
<message name="m1">
  <part name="body" element="tns:GetCompanyInfo"/>
</message>

<message name="m2">
  <part name="body" element="tns:GetCompanyInfoResult"/>
  <part name="docs" type="xsd:string"/>
  <part name="logo" type="tns:ArrayOfBinary"/>
</message>

<portType name="pt1">
  <operation name="GetCompanyInfo">
    <input message="m1"/>
    <output message="m2"/>
  </operation>
</portType>
```

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001
**Bindings and end points**

- A binding defines message formats and protocol details for the operations and messages of a given Port Type (end point in the new spec).

- A binding corresponds to a single end point (obvious since it needs to refer to the operations and messages of the end point).

- An end point can have several bindings (thereby providing several access channels to the same abstract service).

- The binding is extensible with elements that allow to specify mappings of the messages and operations to any format or transport protocol. In this way WSDL is not protocol specific.

- An end point specifies the address of a binding, i.e., how to access the service using a particular protocol and format.

- End points can only specify one address and they should not contain any binding information.

- The end point is often specified as part of a service rather than on its own.
<binding name="b1" type="tns:pt1">
  <operation name="GetCompanyInfo">
    <soap:operation soapAction="http://example.com/GetCompanyInfo"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <mime:multipartRelated>
        <mime:part>
          <soap:body parts="body" use="literal"/>
        </mime:part>
        <mime:part>
          <mime:content part="docs" type="text/html"/>
        </mime:part>
        <mime:part>
          <mime:content part="logo" type="image/gif"/>
          <mime:content part="logo" type="image/jpeg"/>
        </mime:part>
      </mime:multipartRelated>
    </output>
  </operation>
</binding>

<service name="CompanyInfoService">
  <port name="CompanyInfoPort" binding="tns:b1">
    <soap:address location="http://example.com/companyinfo"/>
  </port>
</service>
RPC style vs. Document style

- The style of a SOAP message controls the format of the `<soap:Body>` element:

  - **RPC** style, an extra child element of the Body is added to identify the method to be called. Parameters are listed inside this one.
  - **Document** style, the Body contains an arbitrary XML document.

```xml
<soap:Body>
  <tns:ConfirmOrder xmlns:tns="http://my.package/">
    <number xsi:type="xsd:integer">1234</number>
    <confirm xsi:type="xsd:boolean">true</confirm>
  </tns:Method>
</soap:Body>
```

```xml
<soap:Body>
  <tns:order number="00001234">
    Purchase Order Confirmation
    <tns:status>Confirmed</tns:status>
    ...
  </tns:order>
</soap:Body>
```

```java
ConfirmOrder(number,confirm);
```

```java
Send(OrderDocument);
```
Controlling the style

- The style of the SOAP message is specified in a WSDL document in the binding section.

```xml
<wsdl:binding name="..." type="tns:port type name">  
  <soap:binding style="rpc|document">  
    ...  
  </soap:binding>  
</wsdl:binding>
```

- The style is also reflected in the `<wsdl:message>` element, which can have:
  - For Document style, at most 1 `<wsdl:part element="..."/>
  - For RPC style, any number of `<wsdl:part type="..."/>

- With RPC style you only need an `<wsdl:types>` element to define the complex types used by the parameters.

**Note:** Document style is more general as it can implement the RPC style by using an appropriate XML Schema.
Controlling the encoding

- The serialized data content of a SOAP messages can also be encoded in different ways:
  - Literal (follow the XML schema definition of the WSDL)
  - SOAP encoded (follow Section 5 of the SOAP 1.1 spec)

- The encoding is specified in the WSDL binding section, for each message exchanged as part of each operation:
  ```xml
  <wsdl:input>
    <soap:body use="literal"/>
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </wsdl:input>
  ```

- Style and Encoding are usually paired (Document/Literal and RPC/Encoded). However, they are orthogonal and can be selected independently when the service is deployed.

- These are also orthogonal with respect to the MEP (Message Exchange Pattern) used by the operation.
Services

- Services group a collection of ports together and therefore become the complete definition of the service as seen by the outside:
  - a service supports several protocols (it has several bindings)
  - access to the service under a given protocol is through a particular address (specified in the ports of each binding)
  - the operations and messages to exchange are defined in the End Point

- Ports that are part of the same service may not communicate with each other

- Ports that are part of the same service are considered as alternatives all of them with the same behavior (determined by the End Point) but reachable through different protocols
WSDL example (1)

```xml
<?xml version="1.0"?>
<definitions name="StockQuote" targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns:xsd1="http://example.com/stockquote.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

<message name="GetTradePriceInput">
  <part name="tickerSymbol" element="xsd:string"/>
  <part name="time" element="xsd:timeInstant"/>
</message>

<message name="GetTradePriceOutput">
  <part name="result" type="xsd:float"/>
</message>

<portType name="StockQuotePortType">
  <operation name="GetTradePrice">
    <input message="tns:GetTradePriceInput"/>
    <output message="tns:GetTradePriceOutput"/>
  </operation>
</portType>

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001
```
WSDL example (2)

```xml
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetTradePrice">
    <soap:operation soapAction="http://example.com/GetTradePrice"/>
    <input>
      <soap:body use="encoded" namespace="http://example.com/stockquote"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:body use="encoded" namespace="http://example.com/stockquote"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>
</binding>

<service name="StockQuoteService">
  <documentation>My first service</documentation>
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">
    <soap:address location="http://example.com/stockquote"/>
  </port>
</service>

<definitions>
  From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001
</definitions>
```
4 Universal Description, Discovery and Integration (UDDI)

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
http://www.iks.inf.ethz.ch/
Positioning UDDI

Service Discovery

Messaging

UDDI

Service Interface

Description

SOAP

WSDL

XML

HTTP
Once it is possible to interact with any service provider using the standard SOAP protocol, it is still necessary to:

- Describe the services (WSDL, Web Services Description Language)

- **Discover the services** (UDDI, Universal Description, Discovery and Integration)
Universal Description, Discovery and Integration (UDDI)
What is UDDI?

- The UDDI specification is probably the one that has evolved the most from all specifications we have seen so far. Work on UDDI has ceased
  - version 1 defined the basis for a business service registry
  - version 2 adapted the working of the registry to SOAP and WSDL
  - version 3 redefined the role and purpose of UDDI registries, emphasizes the role of private implementations, and deals with the problem of interaction across private and public UDDI registries

- Originally, UDDI was conceived as an “Universal Business Registry” similar to search engines (e.g., Google) which will be used as the main mechanism to find electronic services provided by companies worldwide. This triggered a significant amount of activity around very advanced and complex scenarios (Semantic Web, dynamic binding to partners, runtime/automatic partner selection, etc.)

- Nowadays UDDI is far more pragmatic and recognizes the realities of B2B interactions: it presents itself as the “infrastructure for Web services”, meaning the same role as a name and directory service (i.e., binder in RPC) but applied to Web services and mostly used in constrained environments (internally within a company or among a predefined set of business partners)
Role of UDDI

- Services offered through the Internet to other companies require much more information than a typical middleware service.
- In many middleware and EAI efforts, the same people develop the service and the application using the service.
- This is obviously no longer the case and, therefore, using a service requires much more information than is typically available for internal company services.
- This documentation has three aspects to it:
  - basic information
  - categorization
  - technical data
More detailed (ebXML architecture)
Information in an UDDI registry
An entry in an UDDI registry is an XML document composed of different elements (labeled as such in XML), the most important ones being:

- **businessEntity**: is a description of the organization that provides the service.
- **businessService**: a list of all the Web services offered by the business entity.
- **bindingTemplate**: the technical aspects of the service being offered.
- **tModel**: (“technical model”) is a generic element that can be used to store additional information about the service, typically additional technical information on how to use the service, conditions for use, guarantees, etc.

Together, these elements are used to provide:

- **white pages information**: data about the service provider (name, address, contact person, etc.)
- **yellow pages information**: what type of services are offered and a list of the different services offered
- **green pages information**: technical information on how to use each one of the services offered, including pointers to WSDL descriptions of the services (which do not reside in the UDDI registry)
Business entity

- The generic white and yellow pages information about a service provider is stored in the businessEntity, which contains the following data:
  - each businessEntity has a businessKey
  - discoveryURLs: a list of URLs that point to alternate, file based service discovery mechanisms.
  - Name: (textual information)
  - Business description: (textual information)
  - Contacts: (textual information)
  - businessServices: a list of services provided by the businessEntity
  - identifierBag: a list of external identifiers
  - categoryBag: a list of business categories (e.g., industry, product category, geographic region)

- The businessEntity does not need to be the company. It is meant to represent any entity that provides services: it can be a department, a group of people, a server, a set of servers, etc
Business service

- The services provided by a business entity are described in business terms using businessService elements.
- A businessEntity can have several businessServices but a businessService belongs to one businessEntity.
- The businessService can actually be provided by a different businessEntity than the one where the element is found. This is called projection and allows to include services provided by other organizations as part of the own services.
- It contains:
  - a serviceKey that uniquely identifies the service and the businessEntity (not necessarily the same as where the businessService is found)
  - name: as before
  - description: as before
  - categoryBag: as before
  - bindingTemplates: a list to all the bindingTemplates for the service with the technical information on how to access and use the service.
A binding template contains the technical information associated to a particular service. It contains the following information:

- bindingKey
- serviceKey
- description
- accessPoint: the network address of the service being provided (typically a URL but it can be anything as this field is a string: e.g., an e-mail address or even a phone)
- tModels: a list of entries corresponding to tModels associated with this particular binding. The list includes references to the tModels, documents describing these tModels, short descriptions, etc.
- categoryBag: additional information about the service and its binding (e.g., whether it is a test binding, it is on production, etc)

A businessService can have several bindingTemplates but a bindingTemplate has only one businessService

The binding template can be best seen as a folder where all the technical information of a service is put together
A tModel is a generic container of information where designers can write any technical information associated to the use of a Web service:

- the actual interface and protocol used, including a pointer to the WSDL description
- description of the business protocol and conversations supported by the service
- “any concept that is not better represented by one of the other UDDI data structures”.

A tModel is a document with a short description of the technical information and a pointer to the actual information. It contains:

- tModelKey
- Name & Description
- overviewDoc: (with an overviewURL and useType that indicate where to find the information and its format, e.g., “text” or “wsdldescription”)
- identifierBag & categoryBag

A tModel can point to other tModels and eventually different forms of tModels will be standardized (tModel for WSDL services, tModels for EDI based services, etc.)
Summary of the UDDI data model

**BusinessEntity**
- businessKey, name, contact, description, identifiers, categories

**BusinessService**
- serviceKey, businessKey, name, description, categories

**BindingTemplate**
- bindingKey, serviceKey, description, categories, access point

**WSDL Document**
- External Web Service Interface Description (located at the service provider)

**tModel**
- name, description, overview document, url pointer to WSDL
Interacting with a UDDI registry
Inquiry and Publishing interfaces

- Access to an UDDI registry typically takes place through SOAP messages that are used to invoke the corresponding API.
UDDI interfaces

- The UDDI specification provides a number of Application Program Interfaces (APIs) that provide access to an UDDI system:
  - UDDI Inquiry: to locate and find details about entries in an UDDI registry. Support a number of patterns (browsing, drill-down, invocation)
  - UDDI Publication: to publish and modify information in an UDDI registry. All operations in this API are atomic in the transactional sense
  - UDDI Security: for access control to the UDDI registry (token based)
  - UDDI Subscription: allows clients to subscribe to changes to information in the UDDI registry (the changes can be scoped in the subscription request)
  - UDDI Replication: how to perform replication of information across nodes in an UDDI registry
  - UDDI Custody and Ownership transfer: to change the owner (publisher) of information and ship custody from one node to another within an UDDI registry

- UDDI also provides a set of APIs for clients of an UDDI system:
  - UDDI Subscription Listener: the client side of the subscription API
  - UDDI Value Set: used to validate the information provided to an UDDI registry
UDDI inquiry API

- Search and lookup entries in a registry.
- This API is freely available, no client authentication is required.
- Errors are reported as SOAP Faults.
- Browse functions search the registry based on keywords and return summary lists with overview information (key, name and description) about matching businesses or services.
- Find qualifiers are used to sort the results and to control the keyword matching: toggle between AND/OR, case sensitive/insensitive, use of wildcards and categories.
- To minimize the number of requests, find queries can be nested.

Drill-down functions are used to fetch the specific UDDI data structures about particular entries given their key, returned by the Browse functions.

<table>
<thead>
<tr>
<th>Browse functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>find_business</td>
</tr>
<tr>
<td>find_relatedBusinesses</td>
</tr>
<tr>
<td>find_service</td>
</tr>
<tr>
<td>find_binding</td>
</tr>
<tr>
<td>find_tModel</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drill down functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_businessDetail</td>
</tr>
<tr>
<td>get_operationalInfo</td>
</tr>
<tr>
<td>get_serviceDetail</td>
</tr>
<tr>
<td>get_bindingDetail</td>
</tr>
<tr>
<td>get_tModelDetail</td>
</tr>
</tbody>
</table>

UDDI Version 3.0 Specification, 19 July 2002
UDDI publishing and security API

- Publish, update and delete information contained in an UDDI registry
- The publishing API requires user authentication using a session token and typically uses SOAP over HTTPS
- The registry performs access control for all publishing functions: information about the entries can only be edited by the owner
- Category information and keyed references associated to the entries are validated before accepting new information into the registry
- Deletion functions are used to remove entries identified by their key from the registry. Removing a business will remove all services associated with it.

The same publishing functions are used both to add new information or replace existing information, depending on whether a valid key is passed or not. When adding new entries, keys are usually automatically generated by the registry.

Security Session Management

- get_authToken, discard_authToken

Publishing

- save_business
- save_service
- save_binding
- save_tModel

Deletion

- delete_business
- delete_service
- delete_binding
- delete_tModel
Summary UDDI

- The UDDI specification is rather complete and encompasses many aspects of an UDDI registry from its use to its distribution across several nodes and the consistency of the data in a distributed registry.

- Most UDDI registries are private and typically serve as the source of documentation for integration efforts based on Web services.

- UDDI registries are not necessarily intended as the final repository of the information pertaining Web services. Even in the “universal” version of the repository, the idea is to standardize basic functions and then built proprietary tools that exploit the basic repository. That way it is possible to both tailor the design and maintain the necessary compatibility across repositories.

- While being the most visible part of the efforts around Web services, UDDI is perhaps the least critical due to the complexities of B2B interactions (establishing trust, contracts, legal constrains and procedures, etc.). The ultimate goal is, of course, full automation, but until that happens a long list of problems need to be resolved and much more standardization is necessary.
Limitations of UDDI
**Hype and reality**

- There were a few universal UDDI registries in operation (maintained by IBM, Microsoft, SAP, etc).
- These registries were very visible and often the first thing one saw of Web services.
- Most of the entries in them did not work or did not correspond to any real service.
- This has been a source of criticism to Web services in general. The criticism has not been entirely undeserved but it is often misguided: what was there to criticize was not UDDI itself but the use that was been made of it and the hype around dynamic Web services.

- UDDI is rather useful if seen as supporting infrastructure for Web services in well defined and constrained environments (i.e., without public access and where there is a context that provides the missing information).
- Most of the UDDI registries in place today are private registries operating inside companies (recall that the widest use of Web services today is for conventional EAI) or maintained by a set of companies in a private manner.
- UDDI has now become the accepted way to document Web services and supply the information missing in WSDL descriptions.
UDDI Public Registries

- Former UDDI Business Registry (UBR) nodes:
  - **IBM**
    - Inquiry API:
    - Publish API:
      - [https://uddi.ibm.com/ubr/publishapi](https://uddi.ibm.com/ubr/publishapi)
  - **SAP**
    - Inquiry API:
    - Publish API:
      - [https://uddi.sap.com/uddi/api/publish](https://uddi.sap.com/uddi/api/publish)
  - **Microsoft**
    - Inquiry API:
      - [http://uddi.microsoft.com/inquire](http://uddi.microsoft.com/inquire)
    - Publish API:
      - [https://uddi.microsoft.com/publish](https://uddi.microsoft.com/publish)
  - **NTT**
    - Inquiry API:
      - [http://www.uddi.ne.jp/ubr/inquiryapi](http://www.uddi.ne.jp/ubr/inquiryapi)
    - Publish API:
      - [https://www.uddi.ne.jp/ubr/publishapi](https://www.uddi.ne.jp/ubr/publishapi)

- The public UDDI Business registries provided by IBM, Microsoft and SAP have been discontinued since January 2006 ([http://uddi.microsoft.com/about/FAQshutdown.htm](http://uddi.microsoft.com/about/FAQshutdown.htm))

- Since their launch in Sept. 2000, they accumulated over 50’000 service registration entries.
Syntax and Semantics

- The problem with UDDI is the initial ambitious goals:
  - the “Google” of web services (and standardized)
  - automatically find business partners worldwide
  - find the interface and build the application on the fly

- In reality:
  - Nobody does business with partners found at random in the Internet
  - Contract and SLAs are more than syntax, the legal aspect takes precedence
  - Trust and knowing the partner are very important in practice
  - The interface describes the syntax, conversations and more complex functions are not yet sufficiently standardized
  - Most of the information in an UDDI repository is redundant if vertical standardization takes place
UDDI vs. directory services

- UDDI good ideas:
  - Services
  - Standardization
  - Different types of information to describe a service

- However, for SOA within a company, UDDI is not good enough:
  - No role base access
  - No life cycle management
  - No support for advanced features (governance)

- Products have started to deviate from the standard (IBM’s WebSphere)