Universal Description Discovery and Integration (UDDI)

Reference

1. Web Services, Alonso, Springer

Players

- Ariba, IBM, Microsoft.
- The first specifications appeared in September, 2000
- The project gained momentum, combining efforts of more than 300 companies.
- After the release of version 3 of the specification (July, 2002), the control of the UDDI project was handed over to OASIS, a formal, independent standards organization.
- OASIS will determine the next steps for the UDDI specification.

Goals of UDDI

- The primary goal of UDDI is the specification of a framework for describing and discovering Web services.
- The core of UDDI resolves around the notion of business registry, which is essentially a sophisticated naming and directory service.
- UDDI defines data structures and APIs for publishing service descriptions in the registry and for querying the registry to look for published descriptions.
- UDDI APIs are also specified in WSDL with SOAP binding, so that the registry can itself be accessed as a Web service, its characteristics can be described in the registry itself, just like any other Web service.

Goals of UDDI

UDDI registry specifications have two main goals with respect to service discovery:

- First, to support developers in finding information about services, so that they know how to write clients that interact with those services.
- Second, to enable dynamic binding, by allowing clients to query to the registry and obtain references to services of interest.
- UDDI leaves room for both structured and unstructured information.

Goals of UDDI

- UDDI also addresses the notion of Universal Business Registry (UBR). The initial goal of the consortium was supporting worldwide directories where everybody could publish service description (essentially for free) and where everybody could query these directories to find services of interest.
- The idea was that UBRs could contain every service ever deployed in the world.
- With time, the specifications evolved in the direction of also supporting private registry implementations, in conjunction with the interaction of private and public implementations.

Information in UDDI Registry

- A simple way of categorizing the information contained in a UDDI registry is in terms of what each type of information is used for. This categorization is best understood by means of an analogy with the telephone directory:
- **White pages:** They are listings of organizations, of contact information (e.g., telephone or e-mail address), and of the services these organizations provide. Using the registry as a white pages catalogues, UDDI clients can find Web services provided by a given business.

Information in UDDI Registry

- **Yellow pages:** They are classifications of both companies and Web services according to taxonomies that can be either standardized or user-defined.
- Through yellow pages, it is possible to search for services based on the category they belong to, according to a given classification scheme.

Information in UDDI Registry

- **Green pages:** This information described how a given Web service can be invoked. It is provided by means of pointers to service description documents, typically stored outside the registry (e.g. at the service provider's site).

UDDI Data Structures

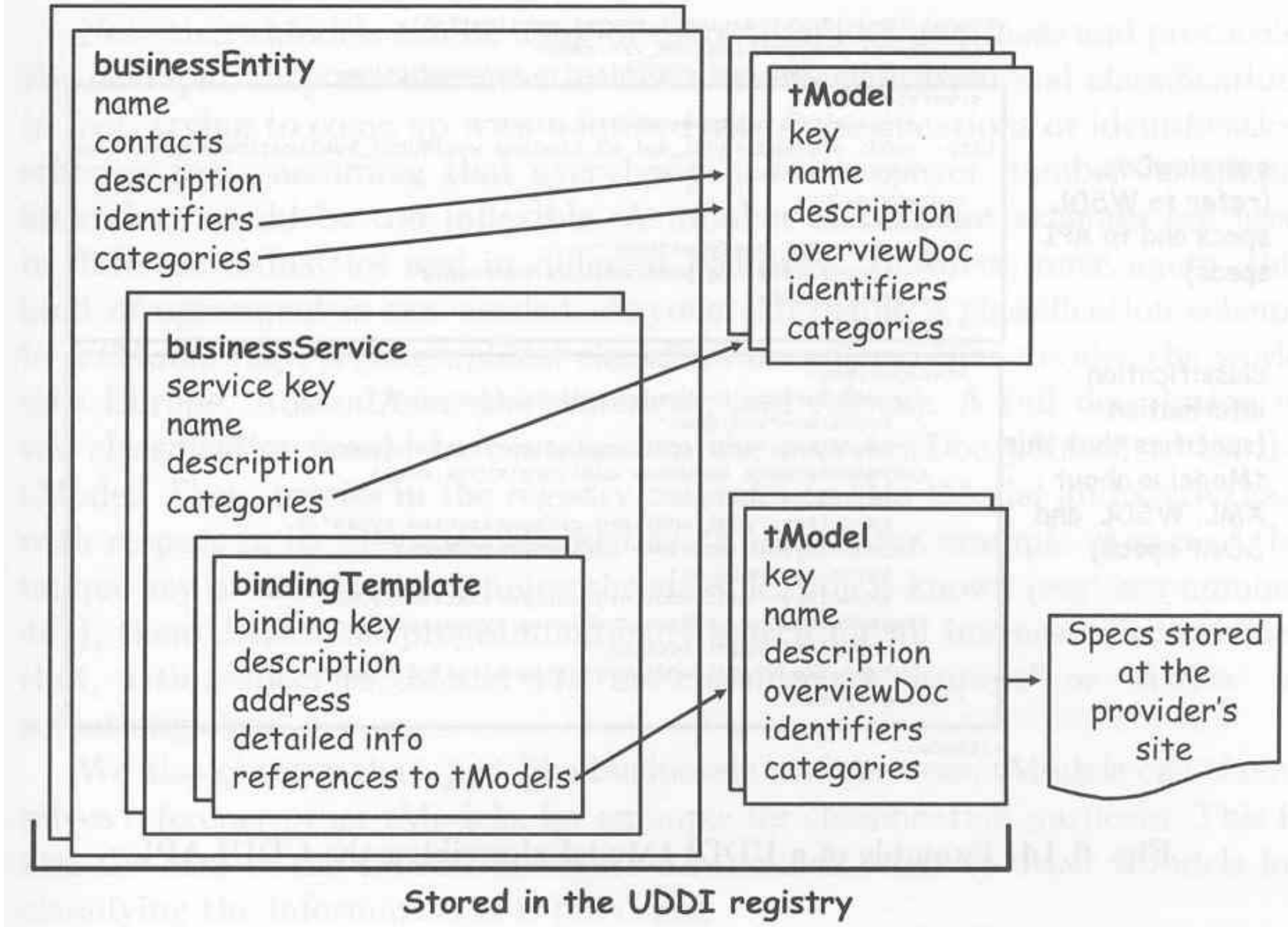
- In the UDDI registry, Web service descriptions contain four types of information:

1. Business information
2. Service information
3. Binding information and
4. Information about specifications of services, described by several entities.

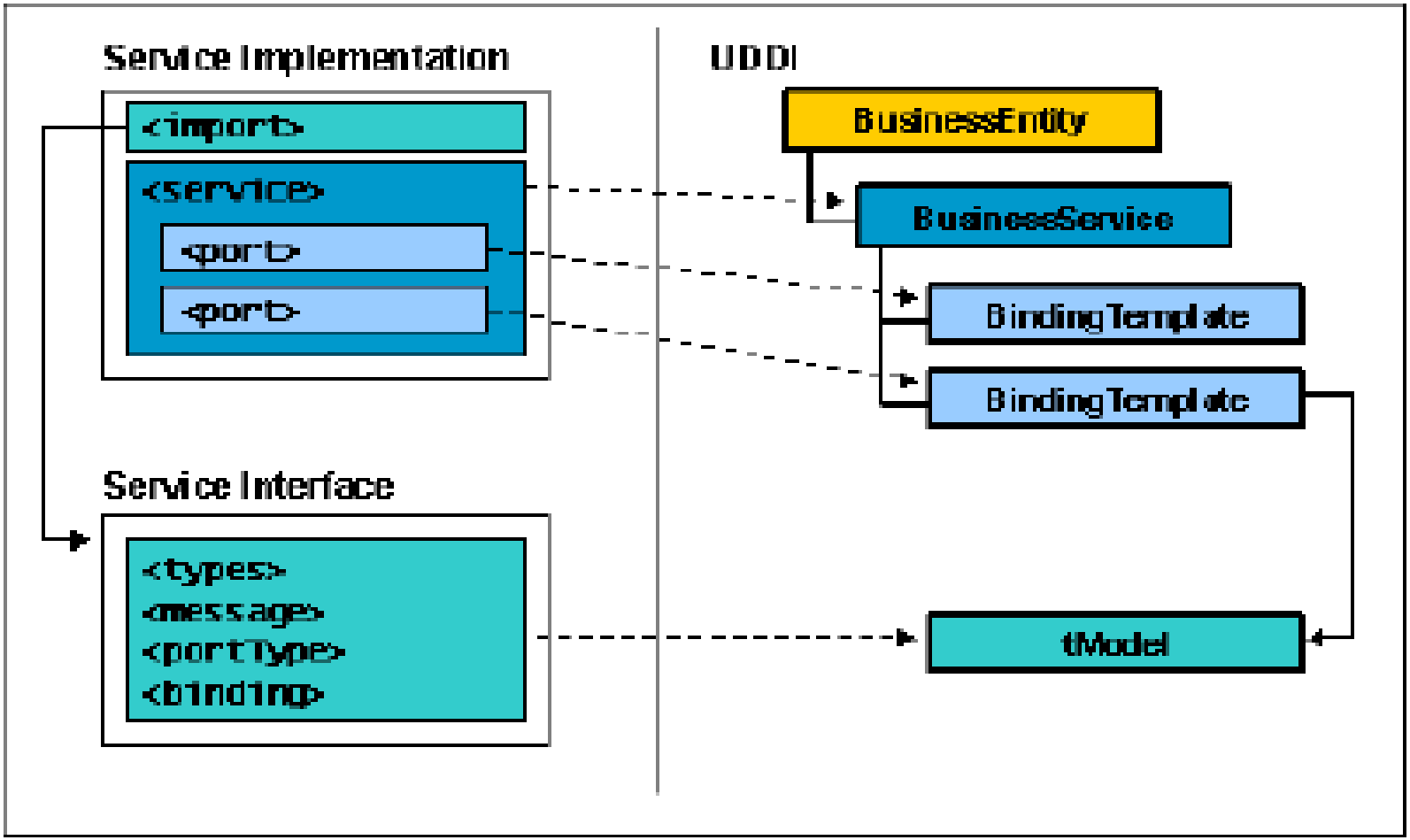
UDDI Data Structures

The four main entities of the registry:

- **businessEntity:** It describes an organization that provides Web services. It lists the company's name, address, and other contact information
- **businessServices:** This element describes a group of related Web services offered by a businessEntity. Typically, a businessService will correspond actually to one kind of services (such as procurement or a travel reservation service), but provided at different addresses, in multiple versions, and through different technologies (e.g. different protocol bindings). Business services, like business entities, can also include classification information.
- A businessEntity entry can include multiple businessService elements, but a businesService belongs to one and only one businessEntity.



A schematic view of a UDDI registry entry



Overview of WSDL to UDDI mapping

UDDI Data Structures

- **bindingTemplate:** This element describes the technical information necessary to use a particular Web service. Essentially, it defines the address at which the Web service is made available along with a set of detailed information, such as references to documents (called tModels) describing the Web service interface or other service properties.
- It also defines how operation parameters should be set and the default values for such parameters. A businessService entry can include multiple bindingTemplate elements (one for each interface or address), but a bindingTemplate belongs to one and only one businessService.

UDDI Data Structures

- **tModel:** The cryptic name stands for “technical model”, and it is a generic container for any kind of specification. For example, it could represent a WSDL service interface, a classification, or an interaction protocol, or it could describe the semantics of an operation. Unlike the previous entities, tModels are not subject to parent-child relationships, and can be referred to by other entities to denote conformance to an interface, or to classify businesses or services.

UDDI Data Structures

- In summary, if we want to publish one or more services in a UDDI registry, we first need to define a set of tModels describing the different characteristics of the services (such as WSDL service interface or an informal description of what the service does).
- Note that some of tModels may be pre-existing, for example in those cases where our services implement standardized interfaces, and the corresponding tModels have been defined and published by the corresponding standardization consortium.

UDDI Data Structures

- We then publish our company information (a businessEntity), general information about the services we provide (a set of businessServices), and a set of detailed technical information for each different implementation and access point we make available for the service (a bindingTemplate). The technical descriptions will contain references to the previously published tModels.

Understanding tModels

- The tModel and the way it is used by the businessEntity, businessService, and bindingTemplate elements, is really at the heart of the UDDI specification. The key information about a service is specified as a reference to a tModel.
- The structure of tModel is shown in the next figure.
- It describes the UDDI API, and specifically the operations provided to publishers in order to advertise their services. UDDI is itself a Web service, and as such it can be described in a UDDI registry and has a WSDL interface.
- The actual content (the service description) resides within a document (the overviewDoc) to thich the tModel points, but that is stored outside the registry, typically at the publishers' site.

overviewDoc (refer to WSDL specs and to API specs)

classification information (specifies that this tModel is about XML, WSDL, and SOAP specs)

```
<tModel tModelKey="uddi:uddi.org:v3_publication">
  <name>uddi-org:publication_v3</name>
  <description>UDDI Publication API V3.0</description>
  <overviewDoc>
      <overviewURL useType="wsdlInterface">
http://uddi.org/wsdl/uddi_api_v3_binding.wsdl#UDDI_Publication_SoapBinding
      </overviewURL>
  </overviewDoc>
  <overviewDoc>
      <overviewURL useType="text">
         http://uddi.org/pubs/uddi_v3.htm#PubV3
      </overviewURL>
  </overviewDoc>
  <categoryBag>
      <keyedReference keyName="uddi-org:types:wsdl"
         keyValue="wsdlSpec"
         tModelKey="uddi:uddi.org:categorization:types"/>
      <keyedReference keyName="uddi-org:types:soap"
         keyValue="soapSpec"
         tModelKey="uddi:uddi.org:categorization:types"/>
      <keyedReference keyName="uddi-org:types:xml"
         keyValue="xmlSpec"
         tModelKey="uddi:uddi.org:categorization:types"/>
      <keyedReference keyName="uddi-org:types:specification"
         keyValue="specification"
         tModelKey="uddi:uddi.org:categorization:types"/>
  </categoryBag>
</tModel>
```

UDDI tModel, describing the UDDI API

Understanding tModels

- In principle, a tModel (or, rather, the overviewDoc refers to) can represent anything and can be written in any language. For example, one could define a tModel that describes an interface for procurement service, and could do so by referring to an overviewDoc that contains specifications in natural language, WSDL, or in any other language for describing interfaces.
- In the context of Web services, of course, the idea is that the service description is done in WSDL.
- The tModel in previous figure includes two overviewDocs, one referring to the WSDL specifications and another containing a textual description of the API.

Understanding tModels

- The content of the overviewDoc, being unstructured, is typically meant to be read by human beings, and not by automated system.
- Once a tModel is published with a UDDI registry, it is assigned a unique key.
- This simple feature, together with the fact that many UDDI entries, possibly belonging to different services or businesses, may refer to the same tModel, is crucial to supporting both developers (design-time discovery) and clients (run-time binding).
- Users can browse a UDDI registry, find services, examine the references to tModels (expressed through the unique keys embedded into bindingTemplate elements), read the tModels, and gain insights on what the service does, what is the interface, and whatever other information is included in the tModel. In this way, developers can understand how to develop client applications.

Understanding tModels

- Assume that a developer knows that a certain tModel, say tModel number 327, describes an interface for supply chain services, as defined by a supply chain standardization consortium. Then, not only can the developer read the tModel to figure out how to write the client application, but he/she can also program the client so that it dynamically binds to a service that supports this standard interface; all that is needed is to program the client so that it searches a UDDI registry for services whose bindingTemplate references tModel 327.
- UDDI does not impose languages for describing interfaces, properties, or semantics. This is left to other standardization consortia, and is considered irrelevant from a discovery and binding descriptions.

Understanding tModels

- Such an open approach enables flexibility and makes UDDI robust to evolution in the Web services scenario. It would simply be impossiblento “hardcode” into the UDDI structure a collection of different standards there are too many and they keep evolving.
- tModels can be used for more than just interfaces and protocols. They ate used in UDDI for identification and classification. In fact, trying to come up with a limited set of classifications or identification schemes (e.g. assuming that everybody has a PAN as unique identified) would be too inflexible.
- A number of different schemes are used in different industries and countries.

Understanding tModels

- Anyone can define a classification scheme in a tModel (say, a geographical classification scheme that divides the world into Europe, Asia, Australia, America, and Africa).
- A full description of the classification would be contained in the overviewDoc pointed to by the tModel. Then, entries in the registry can refer to this tModel and state that, with respect to it, they are classified as “Europe”.
- Again if the unique key of the tModel defining the classification is known (say, key number 441), then clients can programmatically search for all businesses or services that, with respect to tModel 441, are classified as “Europe” or “Africa” or something else.

Understanding tModels

- Just like businesses and services, tModels can themselves reference other tModels, for example for classification purposes. This is also the case of the tModel as shown in previous figure, which refers to other tModels for classifying the information it is providing.
- In addition to allowing custom taxonomies, UDDI also defines a categorization scheme (a specific tModel) that can be used to classify other tModels.

Understanding tModels

- This predefined classification scheme is defined by a tModel called uddi-org:types, and allows a developer to state, for example, that a tModel defines a WSDL interface or a SOAP message. In this way, clients can search for all tModels that are about WSDL descriptions or SOAP messages.

UDDI Registry API

- UDDI registries have three main types of users to whom they expose their API:

1. service providers that publish services,
2. requesters that look for services, and
3. Other registries that need to exchange information.

- These clients can access the registry through six sets of APIs:

UDDI Registry API

- **The UDDI Inquiry API** includes operation to find registry entries that satisfy search criteria and to get overview information about those entities (find_business, find_service, find_binding, find_tModel), and operations that provide details about a specific entity (get_businessDetail, get_serviceDetail, get_bindingDetail, get_tModelDetail). This API is meant to be used by UDDI browser tools that help developers find information and by clients at run-time, for dynamic binding.

UDDI Registry API

- The **UDDI Publisher API** is directed to service providers. It enables them to add, modify, and delete entries in the registry. Examples of operations it supports are save_business, save_service, save_binding and save_tModel for creating or modifying entries and delete_business, delete_service, delete_binding and delete_tModel for removing entries. Upon publication, the registry also assigns unique keys to the newly added entries.

UDDI Registry API

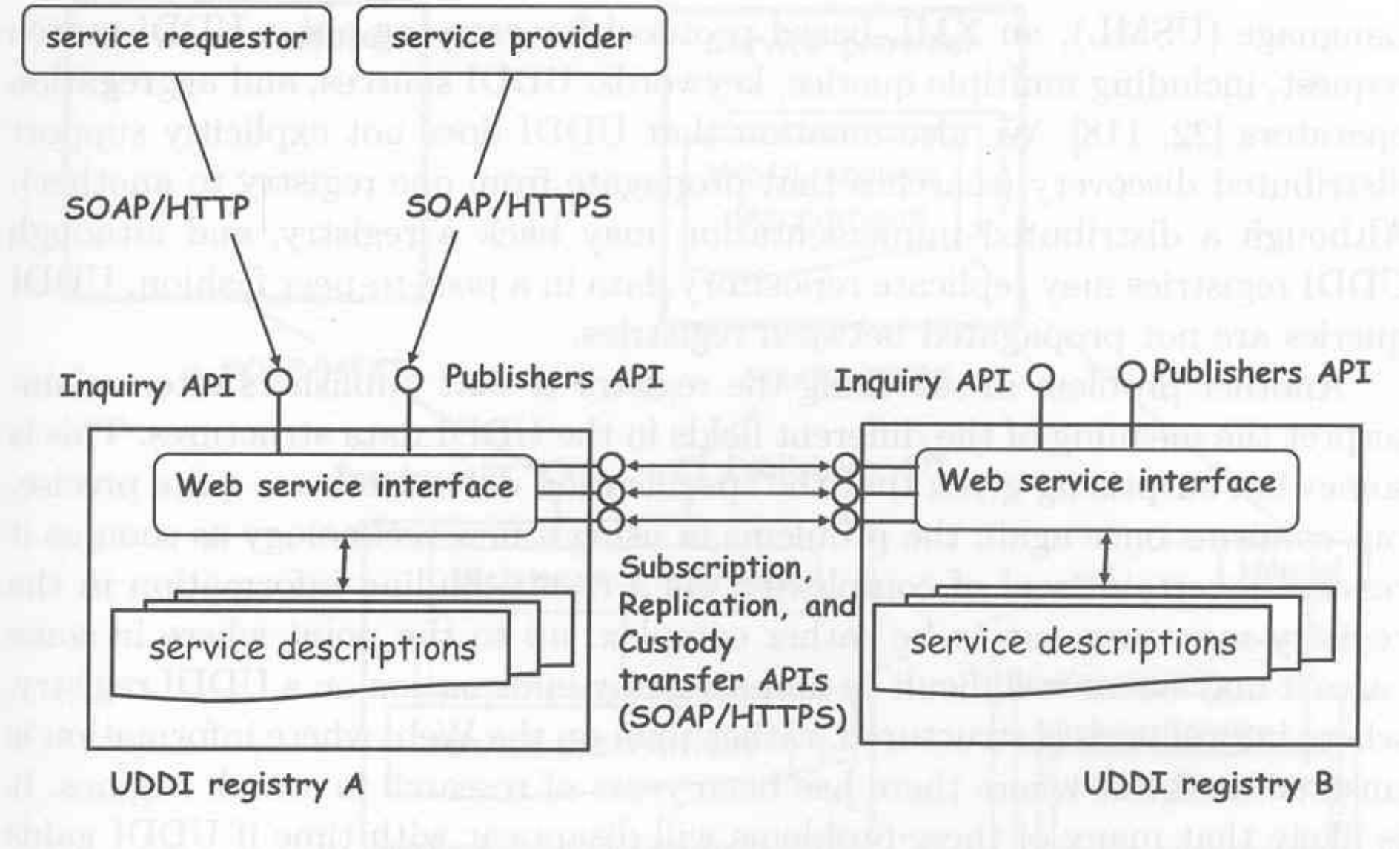
- The **UDDI Security API** allows UDDI users to get and discard authentication tokens to be used in further communication with registry (get_authToken, discard_authToken).
- The **UDDI Custody and Ownership Transfer API** enable registries to transfer the “custody” of information among themselves and to transfer ownership of these structures from one publisher to another. Messages exchanged in this API include get_transferToken, transfer_entities, and transfer_custody.

UDDI Registry API

- The **UDDI Subscription API** enables the monitoring of changes in a registry by subscribing to track new, modified, and deleted entries. The Subscription API includes the following operations: delete_ subscription, get_subscriptionResults, get_subscriptions and save_subscriptions.

UDDI Registry API

- The **UDDI Replication API** supports replication of information between registries, so that different registries can be kept synchronized.
- UDDI registries maintain different access points (URIs) for requesters, publishers and other registries.



Interaction with and between UDDI registries

UDDI Registry API

- Each of these registries exposes an Inquiry API and a Publisher API for service requestors and providers to use, and Subscription, Replication, and Custody Transfer APIs for communicating with other UDDI registries.
- One reason for access point separation is that inquiries do not require authentication (they are standard HTTP) while other invocations require users to authenticate themselves with the registry (thru HTTPS)

UDDI Registry API

- Interaction with a UDDI registry takes place as a series of exchanges of XML documents, typically using SOAP. UDDI specifies the possible message exchanges in terms of requests and responses. It also specifies the structure of the XML documents included in the messages.
- These details are hidden from the developer and user of Web services. Application servers today provide facilities both for publishing services in a UDDI registry and for searching the registry.

UDDI Registry API

- Interaction between registries is needed essentially for two purposes: transferring the custody of an entry and replicating information. Replication occurs according to protocols defined by UDDI, and it is used to keep registries synchronized on either a local or a global scale, depending on the nature of the registry.
- Although the information is replicated, each entry is “owned” by a single registry that, using UDDI terminology, has the custody of the entry. The registry that has the custody is the one the publisher has contarcted when creating the new entry. Modifications to an entry can only be made at the registry that has the custody.

UDDI Registry API

- In some cases, publishers may request to transfer the custody to another registry, perhaps because the previous custodian is ceasing operations or because of contractual agreements. This triggers an inter-registry communication that occurs through Custody Transfer API.

Searching the UDDI Registry

- From a user's perspective, the most important of these APIs is the Inquiry API, which is kept simple.
- Registry providers may offer additional query facilities, perhaps by charging a fee.
- Recently several proposals have appeared: to extend UDDI to support discovery of resources according to characteristics (expected search time) that can change frequently and that are maintained by the service provider.

Searching the UDDI Registry

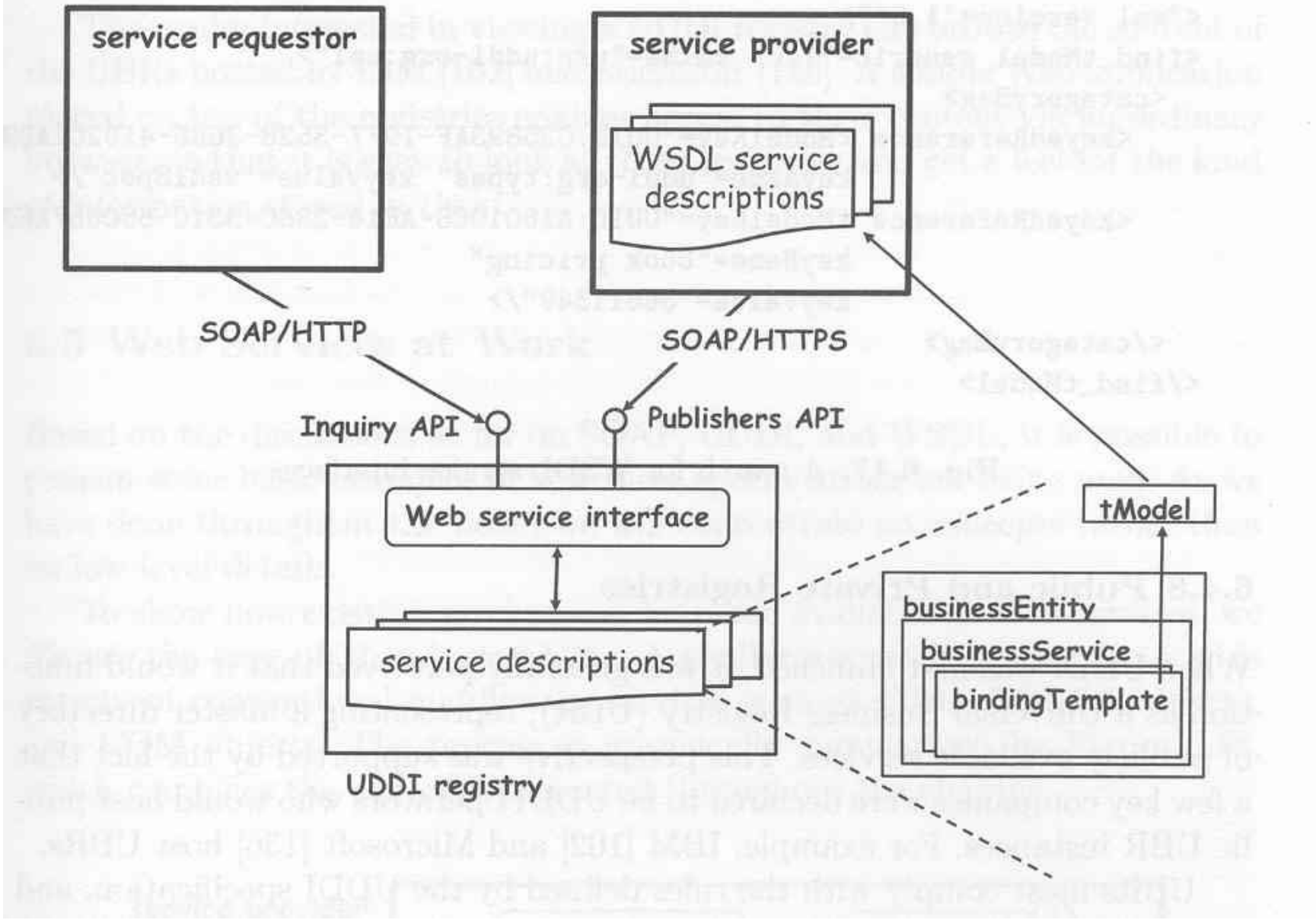
- An initial proposal has been made for the UDDI Search Markup Language (USML), an XML-based protocol for carrying out a UDDI search request, including multiple queries, keywords, UDDI sources, and aggregation operators.
- UDDI does not propagate from one registry to another. UDDI registries may replicate repository data in a P2P fashion, UDDI queries are not propagated between registries.
- Publisher often misinterpret the meaning of the different fields in the UDDI data structures.

Public and Private Registries

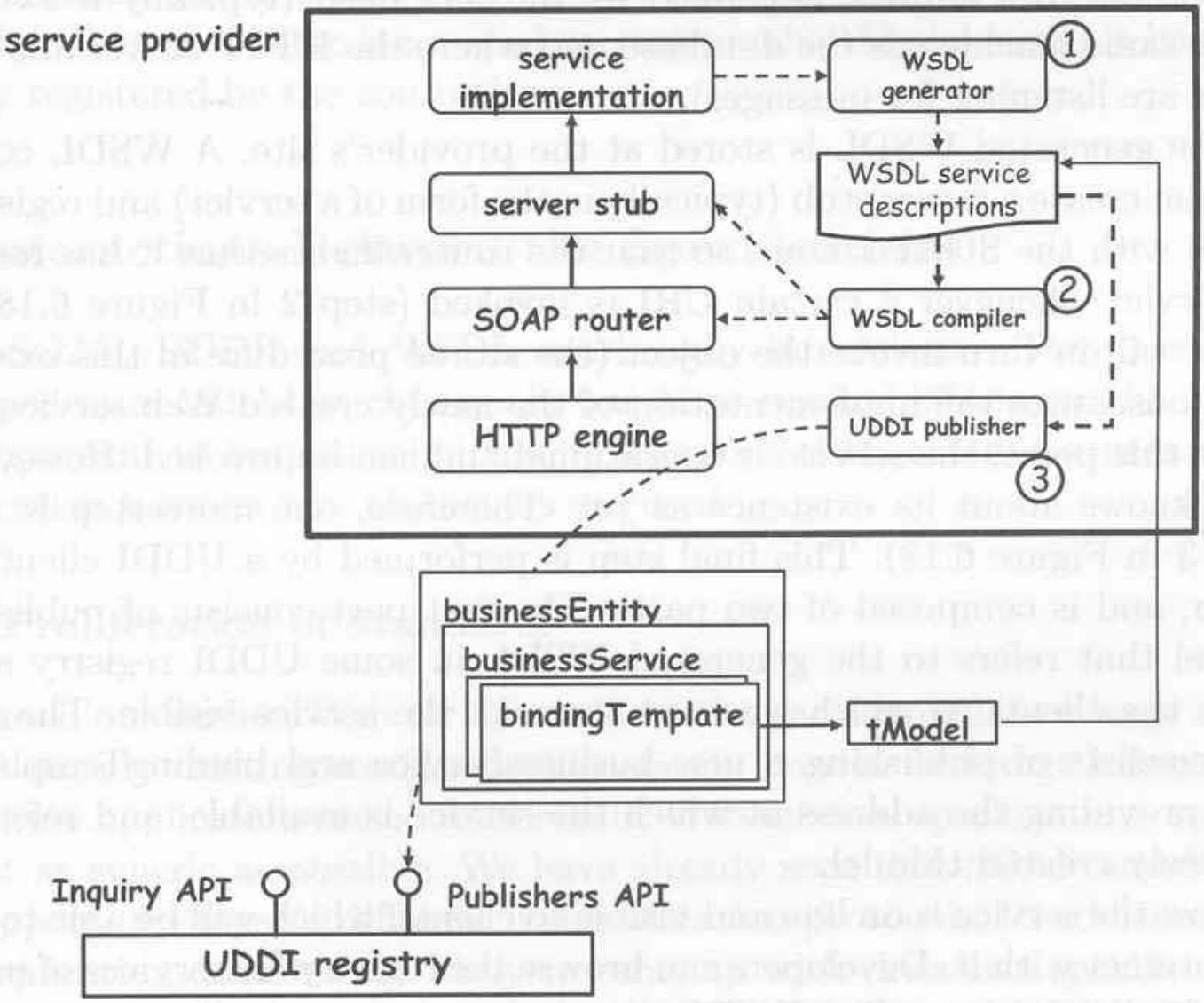
- UDDI was launched to function Universal Business Registry (UBR)
- Few key companies were declared to be UDDI operators who would host public UBRs instaces, e.g. IBM and Microsoft host UBRs.
- UBRs must comply with the rules defines by the UDDI specification, and are supervised by OASIS.
- The content of registry must be kept consistent with the content of other UBRs, and changes in publications must be propagated to the other registries.

Public and Private Registries

- Public registry: provides open and public access to registry data.
- Private registry: an internal registry isolated behind a firewall of a private intranet. An enterprise might use such a registry to support the integration of its own internal applications.
- Shared and semi-private registry: is deployed within a controlled environment and is shared with trusted partners.



A schematic view of the relation between UDDI and WSDL



Exposing and internal service as a Web service