



D10 v0.1 WSMO Registry

WSMO Working Draft 26 April 2004

This version:

<http://www.wsmo.org/2004/d10/v0.1/20040426/>

Latest version:

<http://www.wsmo.org/2004/d10/v0.1/20040426/>

Previous version:

<http://www.wsmo.org/2004/d10/v0.1/20040414/>

Editors:

Reinhold Herzog
Holger Lausen
Dumitru Roman
Peter Zugmann

Authors:

Reinhold Herzog
Holger Lausen
Dumitru Roman
Michael Stollberg
Peter Zugmann

This document is also available in non-normative [PDF](#) version.

Copyright © 2004 [DERI](#)®, All Rights Reserved. [DERI](#) liability, trademark, document use, and software licensing rules apply.

Table of contents

[1. Introduction](#)

[2. Principles and Approach](#)

[3. Architectural Overview](#)

[3.1 Role of Registry in WSMO](#)

[3.2 Retrieval Mechanism](#)

[3.2.1 Syntactical and structural retrieval - UDDI](#)

[3.2.2 Semantic Retrieval - Discovery](#)

[3.2.3 Scalability](#)

[3.2.4 Consistency](#)

[4. Using UDDI in WSMO](#)

[4.1 Supported WSMO Elements](#)

[4.2 Lifecycle of published information](#)

[4.3 UDDI Data Model Configurations](#)

[4.3.1 UDDI Categories Schema](#)

- [4.3.2 UDDI TModel-Keys](#)
- [4.3.3 Mapping WSMO properties into the Registry](#)
- [4.4 Interfacing UDDI](#)
- [4.5 Publishing Interface](#)
 - [4.5.1 Publish WSMO Service](#)
 - [4.5.2 Publish WSMO Ontology](#)
 - [4.5.3 Publish WSMO Goal](#)
 - [4.5.4 Publish WSMO Mediator Service](#)
- [4.6 Retrieval Interface](#)
 - [4.6.1 Retrieval Methods](#)
 - [4.6.2 Retrieval by non-WSMO aware Client](#)
 - [4.6.3 Retrieve WSMO Elements](#)

[References](#)

[Acknowledgements](#)

[Appendix A: Test implementation](#)

Related Publications

- UDDI-V2: <http://www.oasis-open.org/committees/uddi-spec/tcspecs.shtml#uddiv2>
 - UDDI-V3: <http://www.oasis-open.org/committees/uddi-spec/tcspecs.shtml#uddiv3>
-

1. Introduction

This document describes how WSMO modeling elements can be published and retrieved using a Registry. WSMO elements can be retrieved using certain search criteria like business entity or selected WSMO properties.

The Registry is supposed to publish, store and retrieve the following WSMO elements:

- WSMO Services (including WSMO Mediator Services)
- WSMO Ontologies
- WSMO Goals

The document is structured as follows: Section 2 explains the principles and the approach for designing the WSMO Registry; Section 3 introduces the conceptual architecture of the WSMO Registry, and Section 4 specifies the usage of UDDI by defining the mappings for WSMO elements into UDDI technology.

2. Principles and Approach

Requirements for Registry are:

- Have a publish/retrieval procedure for WSMO Services, Goals and Ontologies
- Reuse and adapt existing and well accepted standard methods and tools opposed to specify a new Registry from scratch
- Provide the possibility to extend existing Web Services with WSMO Extensions
- Allow coexistence of existing Web Services and WSMO Services

According to the above criteria, UDDI was chosen as the basis. Major design decision criteria have been:

- Since there is no upper Ontology in WSMO that describes the Registry, it was decided not to define it here, but to use the formalism of UDDI as it is.
- The recommended approach is to store as little information as possible in the Registry itself (the minimum that is required to perform a structural query) and just refer to the WSMO Extensions stored elsewhere.

This document describes how UDDI should be used in WSMO and will show, how properties of WSMO elements can be mapped into the UDDI data model so that UDDI can be used for retrieval using its standard find functions.

A major assumption is that UDDI remains “Ontology agnostic”, i.e. all semantic related processing should be kept separate.

In its specification, [UDDI V3](#) provides mechanisms to extend UDDI’s data model as well as its API. Since there is not real [UDDI V3](#) implementation so far, an approach was taken to get along with the concepts and functions available in [UDDI V2](#).

3. Architectural Overview

3.1 Role of Registry in WSMO

For WSMO, semantic retrieval is required. The approach is to clearly separate Registration, Storage and Discovery:

- The *Registry* stores information about the publisher and selected data about WSMO elements. The Registry is “semantic agnostic” and links to the stored information, which itself contains the semantics. The Registry provides a non semantic retrieval interface.
- WSMO elements are *physically stored* according to a P2P approach (in most cases at the owner/provider of a WSMO element).
- *Discovery* has “semantic knowledge” to discover WSMO elements using the Registry to retrieve the physical elements. Discovery provides a semantic retrieval interface.

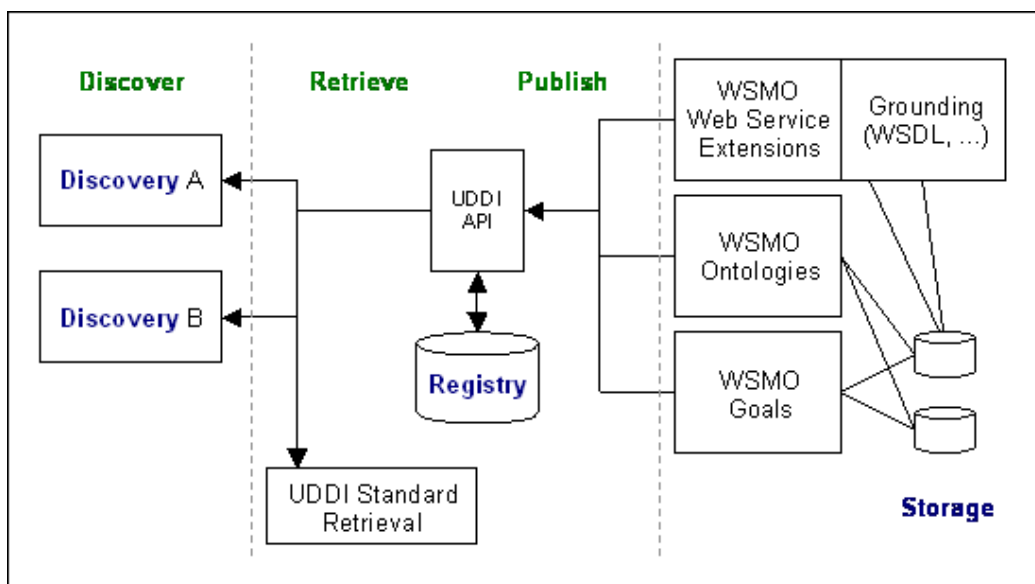


Figure 1. WSMO Architecture for Registry, Discovery and Storage

Since Discovery is key to WSMO, it is the driving point for a certain architecture. A two level approach is suggested:

1. Syntactical and structural retrieval (using standard UDDI functions)
2. Semantic retrieval (using Discovery)

The main reasons to separate into two steps are:

- Any number of discovery mechanisms can be provided, independent from the Registry; Discoveries may vary in capabilities, functions and performance. As well, the UDDI can be used by existing applications in a traditional, non semantic way to retrieve WSMO services.
- Reasoners, used by discovery mechanisms, will most probably need a different data organisation, at least in the current stage (there is no scalable Reasoner which is able to reason distributed data across the net)

3.2 Retrieval Mechanism

3.2.1 Syntactical and structural retrieval - UDDI

WSMO elements can be *syntactically* retrieved using UDDI's standard API, taking into account the mapping that has been applied during publishing.

For an initial load from a Registry, the standard UDDI find and get calls will be used. Find qualifiers are used together with the find API to apply filters for technical and non-technical criteria. For filtering information, only the explicitly stored information in the Registry can be used, definitely not the semantic content within the WSMO compliant service description.

3.2.2 Semantic Retrieval - Discovery

The Discovery is responsible for the semantic retrieval of WSMO elements.

Ontology and WSMO related information is referred to by overviewURL information. It is up to the user to understand the content of the referred documents and is transparent for the Registry. Such contents are WSMO Ontology descriptions, WSMO Service descriptions and WSMO goal descriptions.

Like the Discovery, other mechanisms used by the Discovery, e.g. Reasoners, will use the Registry in the same way as described above.

3.2.3 Scalability

The proposal allows centralized and distributed Registries, and it allows specialized Discovery services. Through partitioning, scalability can be achieved.

Nevertheless the overall design may not just count with the numbers, but must take into account the change frequency of the Registry content, especially if goals with short life cycles should be registered. To improve scalability, the UDDI and the Discovery Service may be combined in a way that the UDDI persistence storage get implemented so that it can be directly shared by the Discovery. Using the Registry, for a Discovery mechanism it will definitely be better to retrieve all WSMO elements in bulk from UDDI as opposed to issue a high number of single requests.

3.2.4 Consistency

Another major issue is the consistency and actuality of the Discovery service. It is up to the implementer of a discovery service to choose the strategy which fits best the balance between costs of

requesting updates from the registry and dealing with potentially outdated or missing information.

4. Using UDDI in WSMO

As already stated, it is proposed to clearly separate Storage, Registry and Discovery components in WSMO. In this approach the Registry can be viewed as an “index” to the storage. Although some recommendations are given in this document, it is up to the implementation to decide how much information about the stored data is copied and stored in the Registry itself.

WSMO elements will be published to UDDI using standard functions. It is not necessary to extend the data model of UDDI (which is possible in UDDI V3), in fact necessary WSMO properties are mapped into existing data slots in UDDI (e.g. BusinessEntity) or in customized slots (Identifier bags). This keeps the Registry “semantic agnostic” and is not dependent on version 3 of UDDI.

4.1 Supported WSMO Elements

The WSMO Registry can be used to publish and retrieve the following WSMO elements:

WSMO Services

Both, the WSMO Extensions (Non Functional Properties etc.) and the Grounding (WSDL) can be published to the Registry. Separation of the Grounding from the WSMO extensions makes it possible to extend existing, already published Services with WSMO properties and to stay compatible with standard UDDI functions.

WSMO Mediator Service

WSMO Mediator Services will be treated as normal WSMO Services. However, they will be represented by a separate tModel implementation in UDDI which makes retrieval easier.

WSMO Ontologies

Ontologies can also be published and retrieved.

WSMO Goals

WSMO Goals have a much more “personal” flavour than other WSMO elements. A WSMO Goal in most cases expresses the “wish” of a user and therefore may contain sensitive or confidential data. Nevertheless some of these Goals may be reusable and provided as templates and therefore should be published using the Registry. It has to be taken into considerations that these Goals should not be subject to very frequent changes (especially in UDDI V3, with it’s replication facilities). Non reusable or frequently updated Goals should be stored at the Client side or other locations which is not elaborated here.

It has to be noted, that for Goals retrieval, all rules, limitations and effects regarding visibility and security apply as specified in UDDI.

4.2 Lifecycle of published information

Since information entries in the Registry have some dependencies on each other, special care has to be taken how to keep them consistent (e.g. WSMO Services refer to used Ontologies or Mediators that do not exist any more).

To allow deletion of all WSMO elements, all elements not owned by a specific business should have an identifier with scheme "WSMObusinessIdentifier", a keyName "owner", and a keyValue "WSMOowner".

Ownership in UDDI is different for deletion and authority to change.

Ownership and re-publishing

In principle UDDI allows anyone who can find a key to re-publish anything. In practice only the owner, as established by the 'authInfo' of an object can re-publish it. There are protocols to transfer ownership and recover ownership. It is recommended no change to these protocols. The only clarification is that an 'un-owned' Ontology can be read after the publishing business has been deleted.

Ownership and BusinessEntity deletion

An Ontology is registered as tModel it therefore can then survive the removal of the Business who published the Ontology. As with all UDDI interactions, a publish action may require an 'authInfo' which, depending on the UDDI policy requires a registered Business relationship with the UDDI provider to edit the Ontology.

Published Services, and Mediators are unavailable after the Business Entity is deleted. Published Goals are still available after the Business Entity is deleted. This can be avoided in a future API wrapper.

4.3 UDDI Data Model Configurations

The UDDI standard data model without extensions is used.

4.3.1 UDDI Categories Scheme

UDDI version 2 does not allow to find a service by identifier. It's proposed to add a Categorization scheme of WSMO elements. Import of the schema into the UDDI depends on the implementation of the UDDI.

As a starting point, the categorization scheme should look as follows (format depends on UDDI implementation, the one below is in the Microsoft UDDI format):

Listing 3: UDDI Categorization scheme

```

<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:uddi="urn:uddi-org:api_v2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:uddi-microsoft-com:api_v2_extensions">
  <categorizationSchemes>
  <categorizationScheme checked="false">
    <uddi:tModel tModelKey="uuid:31094d91-214b-4818-91e3-673ce3fa39ea">
      <uddi:name>WSMO Category Classification Scheme</uddi:name>
      <uddi:description>This categorizes WSMO objects as Business, Service,
Ontology, Goal, MediatorService</uddi:description>
      <uddi:overviewDoc>
        <uddi:overviewURL>http://www.wsmo.org/2004/d3/d3.1/v01/index.html
        </uddi:overviewURL>
      </uddi:overviewDoc>
      <uddi:categoryBag>
        <uddi:keyedReference
tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4" keyName="types"
keyValue="categorization"/>
        <uddi:keyedReference
tModelKey="uuid:be37f93e-87b4-4982-bf6d-992a8e44edab" keyName="Browsing intended"
keyValue="1"/>
        </uddi:categoryBag>
      </uddi:tModel>
      <categoryValue keyValue="1" keyName="WSMOobject" isValid="false"
parentKeyValue="" />
      <categoryValue keyValue="1.1" keyName="WSMOcategory" isValid="true"
parentKeyValue="1" />
      <categoryValue keyValue="1.2" keyName="WSMObusiness" isValid="true"
parentKeyValue="1" />

      <!--is a WSDL service with document containing WSMO specific properties-->
      <categoryValue keyValue="1.3" keyName="WSMOservice" isValid="true"
parentKeyValue="1" />

      <!--defines, restricts terms used by WSMO services, their WSDL documents-->
      <categoryValue keyValue="1.4" keyName="WSMOontology" isValid="true"
parentKeyValue="1" />

      <!--goal are accomplished by orchestrating WSMO services, thru mediators -->
      <categoryValue keyValue="1.5" keyName="WSMOgoal" isValid="true"
parentKeyValue="1" />

      <!--mediates between Ontologies,Goals,Services:sub-categories WW,GG,OO,WG-->
      <categoryValue keyValue="1.6" keyName="WSMOmediatorService" isValid="true"
parentKeyValue="1" />

    </categorizationScheme>
  </categorizationSchemes>
</resources>

```

If a registry does not allow new categorizations to be created for WSMO, then some of the effects can be achieved by adding appropriate tModels to all objects.

4.3.2 UDDI TModel-Keys

For WSMO 5 new, required tModels are introduced:

- WSMOservice
- WSMOontology
- WSMOgoal
- WSMOmediatorService
- WSMObusiness

4.3.3 Mapping WSMO properties into the Registry

A meaningful set of WSMO properties have to be mapped into UDDI to meaningful retrieve WSMO elements in UDDI using UDDI's limited find capability.

Due to UDDI's simple data model, the semantic context of the properties will get lost during the mapping.

Mapping Methods

In principal there are several approaches to map WSMO properties into the Registry:

Automatic establish references:

This means establishing references from UDDI to WSDL for grounding and UDDI to WSMO for semantic specifications.

Manual Mapping:

This covers WSDL Groundings from WSMO interface and UDDI bags from WSMO capabilities and Non Functional Properties.

Semi-Automatic Mapping:

As long as there is no specification within WSMO for grounding, a WSDL must always be created on top, therefore there is no automatic or semi-automatic support. UDDI mapping can be semi-automated.

Automatic Mapping

Only for UDDI, not for WSDL.

Mapped Properties

It seems to be a good starting point to map the following WSMO properties into UDDI. Since there is no exact format of WSMO properties available, this is shown on a more abstract level. In the examples below for publishing it is shown in principle, how a WSMO property can be mapped into UDDI-Id-bags.

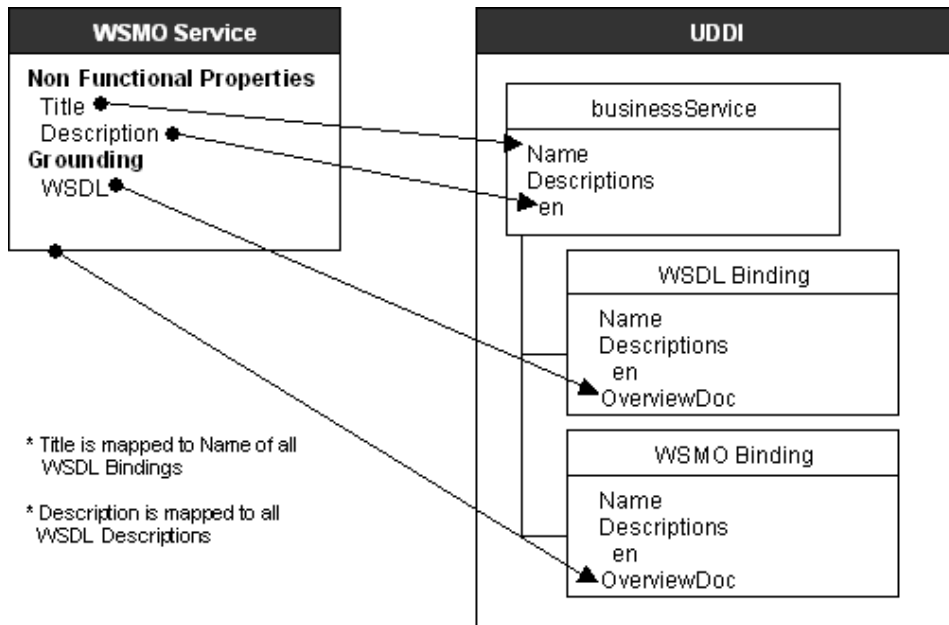


Figure 2. Mapping WSMO Service to UDDI

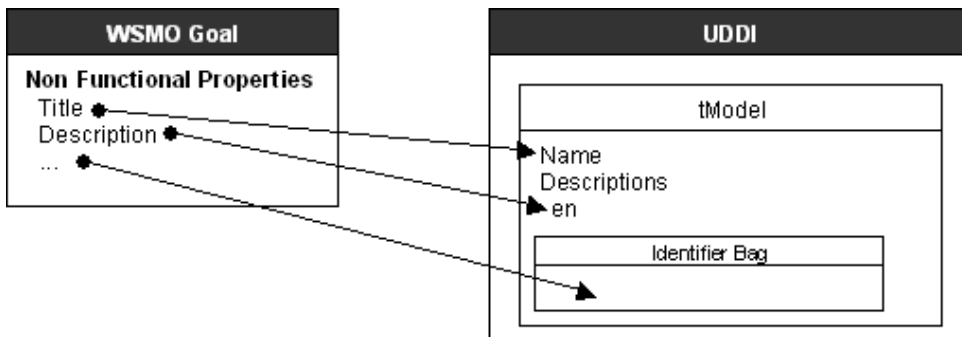


Figure 3. Mapping WSMO Goal to UDDI

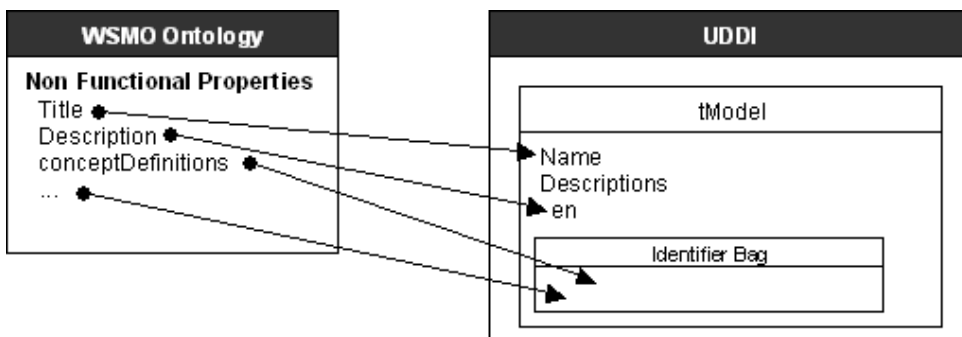


Figure 4. Mapping WSMO Ontology to UDDI

To map WSMO properties into a UDDI bag, a transformation of the property value may be necessary.

For example the NFP “Subject” as described in the WSMO Primer

```
ONFP:/> title ->> {Plane Itinerary Ontology}
        creator ->> {Sinuhe Arroyo}
        subject ->> {travel, leisure, plane}
        ...
]: nonFunctionalProperties
```

would be split into 3 UDDI Identifier bags, each containing for example the key “WSMOnfpSubject” and one of the terms “travel”, “leisure” and “plane”.

In addition to the proposed properties, it is expected that publishers will include additional Category bags and Identifier bags. WSMO will not require any entries, but any publisher expecting their entries to be found directly as well as through Discovery should include as many entries as possible.

4.4 Interfacing UDDI

For a first implementation it is proposed to use native UDDI functions to publish and retrieve WSMO modelling elements.

In a later stage the native API calls can be wrapped into an API (e.g. “WSMO_store” and “WSMO_find”) to provide more convenience for the user and to shield the mapping from the caller. Also, this API could include some validation functions (e.g. check if used Ontologies in a WSMO Service do exist etc.).

Maintenance of the Registry can be done using UDDI’s set of functions.

4.5 Publishing Interface

In the described procedures below it is assumed that a [Categorization scheme](#) as described above has been created in or imported to the UDDI.

4.5.1 Publish WSMO Service

Publishing a WSMO Service in the Registry includes the following steps:

1) Define used Categorization Scheme

Listing 4.5.1.1: Required Category Bag Entry

Category Bag Entry	
Categorization Scheme	"WSMO Category Classification Scheme"
Key Name	"WSMOservice"
Key Value	"1.3"

2) Define the Owner / Publisher of the Service

Owner of a service will be stored in the Identifier bag to be able to retrieve a service by owner. The owner is expected to be the publisher’s name.

Listing 4.5.1.2: Required Identifier Bag Entry

Identifier Bag Entry	
Identification Scheme	"WSMObusinessIdentifier"
Key Name	"owner"
Key Value	Owner name

For each WSMO property mapped into the UDDI, a Identifier bag has to be added. Here is an example for the NFP "language":

Listing 4.5.1.3: WSMO property Identifier Bag Entry

Identifier Bag Entry	
Categorization Scheme	"WSMO Category Classification Scheme"
Key Name	"WSMOservice"
Key Value	"1.3"

Additional Identifier bag entries may be added to be found by non WSMO UDDI usage.

3) Create Binding Template for the Business Service

Listing 4.5.1.4: Binding Template

Binding Template	
AccessPoint:	URL of the WSMO document
Description:	Description from WSMO-core NFPs
Type:	"other"

4) Create tModel

For each Service, a separate tModel must be created. Among other things, this will allow search by Identifier bag.

Listing 4.5.1.5: UDDI tModel for WSMO Service

tModel Properties	
Name:	Title from WSMO-core NFPs
Description:	Description from WSMO-core NFPs
Categories:	Category bag as specified in step 1
Identifiers:	Identifier bag as specified in step 2
Overview Document:	URL of the WSMO Service Extensions

5) Create UDDI Business Service

Listing 4.5.1.6: UDDI Business Service

Business Service Properties	
Service name:	Title from WSMO-core NFPs
Description:	Description from WSMO-core NFPs

6) Add HTTP WSDL Binding

Add HTTP WSDL Binding as normal. This is most important as non-WSMO aware clients will search for, resolve and invoke the service through this binding. This binding must be stand-alone not requiring any WSMO properties, probably this means registering another tModel – and most important cleaning it up. If required, WSMO can assist in cleaning it up if it is assigned a WSMO category and an WSMObusinessIdentifier id has been added to the Identifier bag.

Adding a binding is not described here in detail, but the following data to create the binding should be used:

- URL of WSMO Extension
- Location URL of WSDL location
- Category Bag as specified in step 1
- Identifier bag as specified in step 2

7) *Add Instance for the WSMO Binding*

Listing 4.5.1.7: Instance for WSMO Binding

WSMO Binding	
Interface tModel:	tModel created in step 4
Instance Details:	None
Overview Document:	URL of the WSMO Service Extensions

Re-publishing a WSMO Service is similar to publishing a new service, but requires the Service key, and may require the preserved bags.

4.5.2 Publish WSMO Ontology

Publishing a WSMO Ontology in the Registry includes the following steps:

1) *Define used Categorization Scheme*

Listing 4.5.2.1: Required Category Bag Entry

Category Bag Entry	
Categorization Scheme	“WSMO Category Classification Scheme”
Key Name	“WSMOontology”
Key Value	"1.4"

Additional Category bag entries may be added to be found by non WSMO UDDI usage.

2) *Define the Owner / Creator the Ontology Goal Owner*

The owner of an ontology can be stored in the Identifier bag to be able to retrieve an Ontology by owner. The owner is expected to be the publisher’s name.

Listing 4.5.2.2: Required Identifier Bag Entry

Identifier Bag Entry	
Identification Scheme	“WSMObusinessIdentifier”
Key Name	“owner”
Key Value	Owner name

Additional Identifier bag entries may be added to be found by non WSMO UDDI usage (example see

section [4.5.1.](#))

3) Create tModel

For each Ontology, a separate tModel must be created

Listing 4.5.2.3: UDDI tModel for WSMO ontology

tModel Properties	
Name:	Title from WSMO-core NFPs
Description:	Description from WSMO-core NFPs
Categories:	Category bag as specified in step 1
Identifiers:	Identifier bag as specified in step 2
Overview Document:	URL of the WSMO Ontology

Re-publishing a WSMO Ontology is similar to publishing a new Ontology, but requires the Ontology key, and may require the preserved bags.

4.5.3 Publish WSMO Goal

It's assumed, that Goals must have an Owner. Publishing a WSMO Goal in the Registry includes the following steps:

1) Define used Categorization Scheme

Listing 4.5.3.1: Required Category Bag Entry

Category Bag Entry	
Categorization Scheme	"WSMO Category Classification Scheme"
Key Name	"WSMOontology"
Key Value	"1.4"

Additional Category bag entries may be added to be found by non WSMO UDDI usage.

2) Define the Owner / Creator the Ontology Goal Owner

The owner of a Goal will be stored in the Identifier bag to be able to retrieve a Goal by owner. The owner is expected to be the publisher's name.

Listing 4.5.3.2: Required Identifier Bag Entry

Identification Scheme	"WSMObusinessIdentifier"
Key Name	"owner"
Key Value	Owner name

Additional Identifier bag entries may be added to be found by non WSMO UDDI usage (example see section [4.5.1.](#))

3) Create tModel

For each Goal a separate tModel must be created

Listing 4.5.3.3: UDDI tModel for WSMO ontology

tModel Properties	
Name:	Title from WSMO-core NFPs
Description:	Description from WSMO-core NFPs
Categories:	Category bag as specified in step 1
Identifiers:	Identifier bag as specified in step 2
Overview Document:	URL of the WSMO Goal

Re-publishing a WSMO Goal is similar to publishing a new Goal, but requires the Goal key, and may require the preserved bags.

4.5.4 Publish WSMO Mediator Service

As WSMO Mediators are handled similar to WSMO Services, see section [4.5.1](#)) - WSMOservice has to be replaced with WSMOmediatorService.

4.6 Retrieval Interface

UDDI has some limitations regarding its query interface. Most of the limitations are due to the use of the OR bag qualifiers. Solutions to these limitations are not elaborated in this document.

4.6.1 Retrieval Methods

For retrieval, different methods serve different requirements:

Query

Selects a set of WSMO elements restricted by bags (Identifier bags, Category bags, tModel bags, BusinessEntity). It has to be taken into account, that bags used for retrieval have to use keys and values used during Publishing.

A query has the following input:

- 'authInfo' (mandatory or optional depends on UDDI implementation)
- Category bag (optional)
- Identifier bag (optional)
- tModel bag (optional)
- business URI (optional)

If no input is provided, from a query point of view all WSMO services should be returned, but the actual number depends on the UDDI implementation.

Batch Retrieval

Makes available all WSMO elements. For high volumes, a subscription mechanism would be required, which is not worked out here in detail.

The standard UDDI query interface is used. Use of bags requires knowledge of what properties publishers use in them.

4.6.2 Retrieval by non-WSMO aware Client

As today such a client would retrieve by Category bag, tModel, partial name, and business provider. The client would select http (or fax, e-mail, etc.) binding. The binding would derive instances. The caller would call the WSMO service, unaware of it's WSMO properties and binding.

4.6.3 Retrieve WSMO Elements

Retrieving published WSMO Goals and Ontologies is straight forward using standard UDDI find functions by any combination of name, business, categories, identifiers taking into account the specifications for Publishing.

The same is true for WSMO Services (including Mediator Services) unless it is required to search with identifiers (e.g. WSMO properties mapped into Identifier bags), since UDDI V2 does not allow to search for services by identifiers. The workaround to that is:

1) Select Categorization Scheme (mandatory)

Listing 4.6.3.1: Required Category Bag Entry

Category Bag Entry	
Categorization Scheme	“WSMO Category Classification Scheme”
Key Name	“WSMOservice”
Key Value	"1.3"

Additional Category bag entries may be added.

2) Select Service Owner (optional) Define the Owner / Creator the Ontology Goal Owner

Listing 4.6.3.2: Required Identifier Bag Entry

Identifier Bag Entry	
Identification Scheme	“WSMObusinessIdentifier”
Key Name	“owner”
Key Value	Owner name

3) Search for tModels

UDDI find_tModel function is used, supplying name, Category bag Identifier bag as described above.

For each returned tModel, the tModel-detail contains the WSMO Service Extension URL.

To find the Service, for each tModel-Key, a find_service with the Category bag with exactly one entry as described above must be executed.

5. Conclusion

Although a customized solution for WSMO would provide some benefits, taking UDDI as a starting point for a Registry seems to be a good approach. The main advantage using UDDI as a first implementation for a WSMO Registry is, that a lot of mechanisms needed in a Registry (like access rights, administration, interfaces) are already defined, specified and implemented. Secondly, coexistence with existing Web Services is ensured and existing Web Services can be "upgraded" to WSMO by applying WSMO properties.

Mapping of the WSMO Non Functional Properties into the UDDI data model needs some more investigation and should be finalized as soon as the content and format of the NFPs are fixed and agreed.

For WSMO, UDDI will be used in a very specific manner. It is recommended to provide a wrapper API

that shields the publisher and retriever from dealing with details of the mapping. As well, some validation features could also be added.

References

- [Arroyo and Stollberg, 2004] Arroyo, S.; Stollberg, M.: *WSMO Primer*. WSMO Deliverable D3.1, available at: <http://www.wsmo.org/2004/d3/d3.1/v0.1/>
- [Fensel & Bussler, 2002] D. Fensel and C. Bussler: *The Web Service Modeling Framework WSMF*, Electronic Commerce Research and Applications, 1(2), 2002.
- [Roman et al., 2004] D. Roman, U. Keller, H. Lausen (eds.): *Web Service Modeling Ontology - Standard (WSMO - Standard)*, version 0.2 available at <http://www.wsmo.org/2004/d2/v02/>
- [UDDI] Bellwood, T.; Clément, L.; von Riegen, C. (Ed.): *UDDI Version 3.0.1*. UDDI Spec Technical Committee Specification, Dated 20031014. available at: http://uddi.org/pubs/uddi_v3.htm
- [WSDL] Chinnici, R.; Gudgin, M.; Moreaum, J.-J.; Weerawarana, S. (2003): *Web Services Description Language (WSDL) Version 1.2*. W3C Working Draft 3 March 2003. available at <http://www.w3.org/TR/wsdl20/>.

Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperonto, and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate programme.

The editors would like to thank to all the [members of the WSMO working group](#) for their advice and input to this document.

Appendix A: Sample implementation

This appendix describes how a sample WSMO Registry could be set-up. The approach for the sample implementation is to use open source components for setting-up a standard UDDI Registry and to have a WSMO Proxy component, that implements the convenience API as suggested in chapter 4.4 of this document.

A.1 Used components

For Web Server, Application Server, UDDI Server and database, open source tools have been selected. The following components are part of the sample installation:

Java Runtime

Sun J2SE JDK 1.4.2_04

UDDI Server

jUDDI 0.8.0 with source code changes

For the core of the WSMO Registry, jUDDI has been selected. jUDDI has been developed under a Apache project. Currently there is no production version; the latest version is still beta but is expected to available for production soon. jUDDI runs under an Application Server and uses a relational database for storing Registry entries.

WSMO Proxy

The WSMO Proxy is not a standard component. The purpose of the proxy is to provide the WSMO specific convenience functions for publishing and retrieval.

Application Server

JBoss 3.2.3. jUDDI and the WSMO Proxy run under the Application Server

Relational Database

MySQL 4.0.15, used by jUDDI.

JDBC Driver

MySQL Connector/J 3.0.9-stable (formerly MM.MySQL)

Web Server

Apache Web Server 2.0.44. Connection to the Application Server is provided by the Apache modul mod_jk2 2.0.2.

Client

The client must support SOAP to communicate with the Registry.

Figure A1 shows how the components are related to each other and what protocols are used.

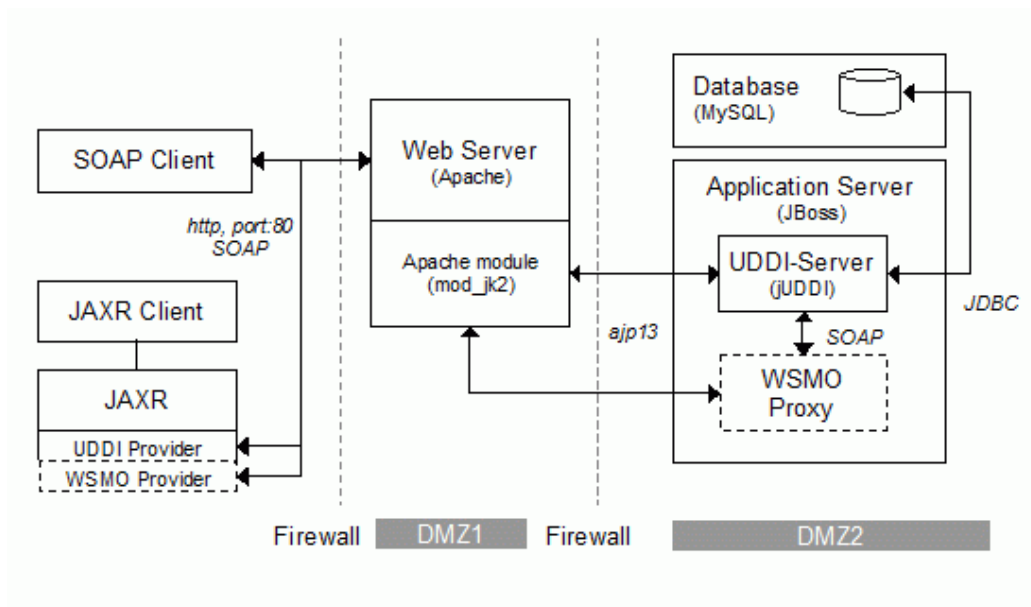


Figure A1. Sample implementation architecture

A test implementation supporting the SOAP interface and standard UDDI functions (without WSMO Proxy) has been set-up at Net Dynamics at www.netdynamics-tech.com/registry.

A.2 WSMO convenience API for UDDI usage

As worked out in this document before, the WSMO Registry should provide a standard UDDI compatible interface so it can be used by clients supporting this interface and it should provide a convenience API that shields the client programmer from dealing with WSMO mappings into UDDI. This functionality will be provided by a WSMO-Proxy.

In the proposed implementation, a WSDL for describing the interface to jUDDI exists (provided with the jUDDI package). The recommendation is to provide as well a WSDL for WSMO-UDDI covering at least the convenience functions required for WSMO. The WSM-UDDI WSDL should also include the most important standard UDDI functions (which will be passed through unchanged) so that the client has to deal with only one WSDL. The less frequently used UDDI functions can be still accessed using the standard UDDI WSDL.

A.3 WSMO Registry client considerations

For accessing the Registry functions there are two suggested options:

- A client that communicates via SOAP with the WSMO Registry using the WSDLs for the standard UDDI Registry and for WSMO convenience functions
- A client that uses JAXR (JAVA API for XML Registries) as an interface to the Registry. JAXR is an API for accessing diverse and heterogeneous business Registries. The advantage of this approach is, that the client does not need to know about the Registry's implementation specifics. So called JAXR Providers link the JAXR functions to the actual Registry. A provider for UDDI already exists. For WSMO it should be checked, if a WSMO Provider can be implemented supporting all required functions.

A.4 Sample installation and configuration

Below the setup and configuration is described as used for the test installation at Net Dynamics. Only the standard UDDI setup is explained here, to apply the WSMO relevant configurations (e.g. categorization schema) please refer to the corresponding chapters above in this document.

MySQL installation and configuration

Download and install MySQL from <http://www.mysql.com/downloads/index.html>.

1. Remove blank user-id to fix the MySQL security gap

```
mysql>DELETE FROM mysql.user WHERE user = '';
```

```
mysql>COMMIT;
```

2. Set root password

```
mysql>UPDATE mysql.user SET password=PASSWORD('rootpassword') -> WHERE user = 'root';
```

```
mysql>COMMIT;
```

```
mysql>FLUSH PRIVILEGES;
```

3. Setup database schema for the registry

Add database user

```
mysql>INSERT INTO mysql.user (host, user, password) -> VALUES('localhost', 'dip',  
PASSWORD('dip'));  
  
mysql>COMMIT;  
  
mysql>FLUSH PRIVILEGES;
```

Create registry database

```
mysql>CREATE DATABASE registry;  
  
mysql>GRANT ALL ON registry.* TO dip@"localhost" IDENTIFIED BY "dip";  
  
mysql>exit
```

Create tables

Remark: juddi_mysql.ddl is part of the jUDDI package

```
prompt>mysql -user=dip -p  
  
mysql>use registry  
  
mysql>source juddi_mysql.ddl  
  
mysql>show tables;
```

Add publisher

The *PUBLISHER_ID* is the user-id the publisher uses for authentication. The *PUBLISHER_NAME* is the UDDI Authorized Name. *ADMIN* indicates if the publisher has administrative privileges.

```
mysql>INSERT INTO PUBLISHER (PUBLISHER_ID,PUBLISHER_NAME,ADMIN) VALUES ('dip','DIP  
User','false');  
  
Mysql>COMMIT;
```

JBoss installation and configuration

Download and install JBoss from <http://www.jboss.org/downloads>

Download MySQL JDBC Connector from <http://dev.mysql.com/downloads/connector/>

1. Configure MySQL DataSource

Create the following XML file and place it in <JBoss install directory>/server/default/deploy/mysql-ds.xml:

Listing A1: MySQL data source configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>Registry</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/registry</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>dip</user-name>
    <password>dip</password>
    <min-pool-size>1</min-pool-size>
    <max-pool-size>20</max-pool-size>
  </local-tx-datasource>
</datasources>
```

2. Install JDBC Driver

Copy mysql-connector.jar into <JBoss install directory>/server/default/lib

jUDDI installation and configuration

jUDDI can be downloaded from <http://sourceforge.net/projects/juddi/>. Unfortunately there are some bugs in jUDDI related to conformance to the UDDI XSD. These bugs have been fixed by Net Dynamics and will be posted to the jUDDI development forum. It is expected that future versions will have this bugs removed. The required source code changes can be download from www.netdynamics-tech.com/registry/download.

1. Deploy jUDDI web application

From the binary distribution or the compiled jUDDI project copy the webapp ./build/webapps/juddi to the <JBoss install directory>/server/default/deploy.

Rename the jUDDI directory to registry.war so it is recognized as a single web application with the context path /registry.

Create the application server specific deployment descriptor file: <JBoss install directory>/server/default//deploy/registry.war/WEB-INF/jboss-web.xml

Listing A2: jUDDI deployment descriptor file

```
<?xml version="1.0" encoding="UTF-8"?>

<jboss-web>

  <resource-ref>

    <res-ref-name>jdbc/juddiDB</res-ref-name>

    <jndi-name>java:/Registry</jndi-name>

  </resource-ref>

</jboss-web>
```

2. Change jUDDI properties

Change the operator name in the properties file <JBoss install directory>/server/default/deploy/registry.war/WEB-INF/classes/juddi.properties into netdynamics-tech.com:

```
# jUDDI Registry Properties (used by RegistryServer)

# see http://www.juddi.org for more information

# The UDDI Operator Name

juddi.operatorName = netdynamics-tech.com
```

3. User configuration

Add a user with the user-id "dip" and password "dip" to the user configuration file in <JBoss install directory>/server/default/deploy/registry.war/WEB-INF/classes/juddi.user:

Listing A3: jUDDI User configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<juddi-users>

  <user userid="dip" password="dip">

    ...

</juddi-users>
```

Apache HTTP Server installation and configuration

Download and install the Apache HTTP Server from <http://httpd.apache.org/download.cgi>.

Download the mod_jk2 module from <http://jakarta.apache.org/site/binindex.cgi>.

1. Install mod_jk2 module

Copy mod_jk2.so to <Apache install directory>/modules.

Insert the line "LoadModule jk2_module modules/mod_jk2.so" into the appropriate section in the file <Apache install directory>/conf/httpd.conf.

2. Configure mod_jk2 module

Create a mod_jk2 configuration file <Apache install directory>/conf/workers2.properties.

Listing A4: mod_jk2 module configuration

```
[shm]

file=${serverRoot}/logs/shm.file

size=1048576

# Example socket channel, override port and host.

[channel.socket:localhost:8009]

port=8009

host=127.0.0.1

# define the worker

[ajp13:localhost:8009]

channel=channel.socket:localhost:8009

# Uri mapping

[uri:/sample/*]

worker=ajp13:localhost:8009

[uri:/RegistryServer/*]

worker=ajp13:localhost:8009

[uri:/registry/*]

worker=ajp13:localhost:8009
```

Replace localhost with your servername where the jBoss Application Server is installed. To activate the new configuration, the Apache Server must be restarted. If the Application server is located behind a Firewall, open the TCP-Port 8009 in the Firewall configuration.

Registry Server start

To start the registry server just use the standard jboss startup script <jBoss installation directory>/bin/run.sh or run.bat if using Windows.

A.5 Sample SOAP Request

The listing below shows a sample SOAP request to find a business.

Listing A5: Sample SOAP request

```
<?xml version="1.0" encoding="UTF-8"?>

<SOAP-ENV:Envelope

  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Header/>

  <SOAP-ENV:Body>

    <find_business generic="2.0" xmlns="urn:uddi-org:api_v2">

      <name xml:lang="en"

        xmlns:xml="http://www.w3.org/XML/1998/namespace">%coff%</name>

      </find_business>

    </SOAP-ENV:Body>

  </SOAP-ENV:Envelope>
```

A.6 Resources

UDDI API - WSDL files

UDDI Inquire API: http://uddi.org/wsd/inquire_v2.wsdl

UDDI Publish API: http://uddi.org/wsd/publish_v2.wsdl

UDDI XML Schema

Version 2.0 UDDI XML Schema 2001: http://uddi.org/schema/uddi_v2.xsd

\$Date: Montag 26 April 2004 - 09:54:24\$

[webmaster](#)