TripCom

Triple Space Communication

FP6 – 027324

Deliverable

D4.5

Triple Space Integration with respect to WSMX

Brahmananda Sapkota (NUIG)
Zhangbing Zhou (NUIG)
Omair Shafiq (LFUI)
Daniel Wutke (USTUTT)

April 27, 2009

Executive Summary

The goal of this deliverable D4.5 - *Triple Space Integration with respect to WSMX* is to present the details about the integration work of Triple Space Computing and WSMX as proposed and discussed in previous deliveries D4.1, D4.2 and D4.3. This deliverable also presents the implemented integration prototype which shows how Triple Space Computing can be used as a Semantic Web Services platform. As part of the integration prototype Web Service Client, Web service Registry, Web service discovery, Web service Invocation, and Resource Management components are implemented.

In particular, this deliverable presents the implementation task from two different but complementary perspectives: (1) *the internal integration* which shows how WSMX internal components can be integrated through Triple Space, and (2) *the external integration* which shows (i) how the client can access various components and use their functionalities to enable external components interact with WSMX via Triple Space and (ii) how WSMX can interact with Web services over Triple Space.

In addition to detailing integration prototype, this deliverable also presents the user guide specifying how to use the presented prototype. Information regarding the licences under which the presented prototype is released, third party dependencies and their respective licensing information are pointed out where necessary.

Document Information

IST Project Number	FP6 – 027324	**Acronym**	TripCom
Full Title	Triple Space Communication		
Project URL	http://www.tripcom.org/		
Document URL			
EU Project Officer	Werner Janusch		

Deliverable	**Number**	4.5	**Title**	Triple Space Integration with respect to WSMX
Work Package	**Number**	4	**Title**	Triple Space and Semantic Web Services

Date of Delivery	**Contractual**	M36	**Actual**	31-Mar-09
Status	version 1.0		final ☒	
Nature	prototype ☒ report ☐ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Brahmananda Sapkota (NUIG), Zhangbing Zhou (NUIG), Omair Shafiq (LFUI), Daniel Wutke (USTUTT)			
Resp. Author	Brahmananda Sapkota		**E-mail**	brahmananda.sapkota@deri.org
	Partner	NUIG	**Phone**	+353 (91) 495053

Abstract (for dissemination)	The goal of this deliverable D4.5 - *Triple Space Integration with respect to WSMX* is to present the details about the integration work of Triple Space Computing and WSMX as proposed and discussed in previous deliveries D4.1, D4.2 and D4.3. This deliverable also presents the implemented integration prototype which shows how Triple Space Computing can be used as a Semantic Web Services platform. The implementation task is presented from two different but complementary perspectives: the internal integration of WSMX components through Triple Space, and the external integration of WSMX and its clients. In addition, the licensing information of the presented prototype, its usage guide and the third party dependencies and their licensing information are presented.
Keywords	Triple Space Computing, WSMX, Web Services, Semantic Web Services, Registry, Discovery, Invocation, Resource Management

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
01/04/2008	1	B. Sapkota and Z. Zhou	Initial outline created
09/04/2008	2	O. Shafiq	Added myself as co-author, and comments to initial outline
16/05/2008	3	O. Shafiq	Updated descriptions of client and resource management in Chapter 2
30/05/2008	4	D. Wutke	Intial version of the WS registry section.
31/05/2008	5	O. Shafiq	More updates in in descriptions of client and resource management in Chapter 2
31/10/2008	6	Z. Zhou	More updates in in descriptions of third-party dependency in Chapter 4
25/02/2009	7	O. Shafiq	Cross checking and finalizing LFUI inputs
01/03/2009	8	D. Wutke	Finalized WS registry section, added description of the TS Web service binding to the invocation section, added corresponding 3rd party libraries.
10/03/2009	9	Z. Zhou	Cross checking and finalizing NUIG inputs
23/03/2009	10	B. Sapkota	Finalise the deliverable
30/03/2009	11	Z. Zhou	Take the comments from the quality assessor
03/04/2009	12	Z. Zhou, D. Wutke and B. Sapkota	Take the comments from the quality controller

Project Consortium Information

Acronym	Partner	Contact
Semantic Technology Institute Innsbruck http://www.sti-innsbruck.at	STI	Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria E-mail: dieter.fensel@sti-innsbruck.at
National University of Ireland, Galway http://www.deri.ie	NUIG	Dr. Laurentiu Vasiliu Digital Enterprise Research Institute (DERI) Galway, Ireland Email: laurentiu.vasiliu@deri.org
University of Stuttgart http://www.iaas.uni-stuttgart.de/	USTUTT	Prof.Dr. Frank Leymann Inst. für Architektur von Anwendungssystemen (IAAS) Stuttgart, Germany E-mail: frank.leymann@informatik.uni-stuttgart.de
Vienna university of Technology http://www.complang.tuwien.ac.at/	TUW	Prof.Dr. eva Kühn Institut für Computersprachen Vienna, Austria E-mail: eva@complang.tuwien.ac.at
Free University Berlin http://www.ag-nbi.de/	FUB	Prof. Dr.-Ing. Robert Tolksdorf AG Netzbasierte Informationssysteme Berlin, Germany E-mail : tolk@inf.fu-berlin.de
Ontotext Lab, Sirma Group Corp. http://www.ontotext.com/	ONTO	Atanas Kiryakov, Vassil Momtchev, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: vassil.momtchev@ontotext.com
Profium OY http://www.profium.com/	Profium	Dr. Janne Saarela Profium OY Espoo, Finland E-mail: janne.saarela@profium.com
CEFRIEL SCRL. http://www.cefriel.it/	CEFRIEL	Davide Cerri CEFRIEL SCRL. Milano, Italy E-mail: cerri@cefriel.it
Telefonica I+D http://www.tid.es/	TID	Noelia Pérez Crespo Telefonica I+D Madrid, España E-mail: npc@tid.es

Table of Contents

List of Abbreviations

API	Application programming interface
HTTP	Hyper Text Transfer Protocol
IRI	Internationalized Resource Identifiers
LGPL	GNU Lesser General Public Licence
OASIS	Organization for the Advancement of Structured Information Standards
ORDI	Ontology Representation and Data Integration
RDF	Resource Description Framework
RDFS	RDF Schema
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
WSMO	Web Service Modeling Ontology
WSML	Web Service Modeling Language
WSMX	Web Service Modeling Execution Environment
XML	Extensible Markup Language

1 Introduction

One of the major goals of WP 4 is to align Semantic Web Services with Triple Space Computing, i.e. to allow Triple Space-based service execution as well as communication and coordination for Semantic Web Services which requires interfaces to allow Semantic Web Services to communicate over Triple Space. The goal of this deliverable is to describe the prototype implementation that integrates Triple Space Computing with WSMX as prescribed by previous WP4 deliveries D4.1, D4.2 and D4.3. In general, the functionalities of the prototype are classified from the following two aspects: the internal integration between WSMX components, and the external environment from the client's perspective.

From the internal integration perspective, this deliverable shows how Triple Space can be used as the back-end storage for resources, such as ontologies, WSML Web services, WSML goals, and WSML mediators, and then, how WSMX components, such as registry, discovery, invocation, and resource management can operate through Triple Space operators. Therefore, we need to provide a technique to translate between the representation format of WSML resource format and Triple Space triples. This translation is achieved by means of the *WSMO4RDF* [1] tool.

From the external environment perspective, this deliverable shows how the functionalities provided by the prototype can be accessed by the external systems: either the end user accesses the components through Triple Space enabled Web service client, or the Web service invocation component invokes the external Web Service end-points over Triple Space. Both of these accesses require another technique to translate between the SOAP message format and the triples as used in Triple Space, which is achieved by means of the *SOAP2RDF* tool.

The reminder of this chapter introduces WSMX in brief. It further defines scope and objectives of the deliverable. Target or expected audience for the integration of Triple Space with WSMX have been identified.

1.1 WSMX: A Short Introduction

The Web Services Execution Environment (WSMX) [2] (`http://www.wsmx.org/`) is an execution environment for dynamic discovery, selection, composition, mediation, invocation and execution management of Semantic Web Services. WSMX is a reference implementation of Web Services Modeling Ontology (WSMO) that acts as conceptual model to describe various aspects of Semantic Web Services. WSMO is based on four major fundamental elements which are web service descriptions, ontologies, goals and mediators. WSMX can achieve a user's goal by dynamically selecting a matching Web service, mediating the data that needs to be communicated to this service and invoking it.

WSMX is developed in participatory open source environment. The open source philosophy has been set from the very beginning, in order to allow a world-wide community-style development, to provide everybody access for development and personal usage to an execution environment for the semantic web. It can be found at SourceForge (`http://sourceforge.net/projects/wsmx`) and is available under license LGPL.

The key modules of WSMX are given as follows:

- WSMX Integration API: The WSMX Integration API is a collection of libraries required for the integration of loosely coupled components with the main WSMX system. Components must implement interfaces from the provided infomodel to make this integration possible.
- WSMX Core: is a release of the compiled core of WSMX, along with a set of mock up components that implement the different interfaces in the WSMX Integration API.
- WSMX Components: is a release of the compiled versions of the current components available for WSMX. The components can be plugged into the WSMX Core.

1.2 Scope and Objectives

The scope of this deliverable is to address how the components in WSMX can be integrated with Triple Space for achieving the internal integration between WSMX components, as well as the external communication with the client or the external Web Service endpoints over Triple Space.

First, to achieve the internal integration between WSMX components, in Section 2, we define the scope and present how the components of Web service registry, Web service discovery, Web service invocation, and resource management are implemented by means of Triple Space.

We next explore the external communication with the external system. In Section 2, we define the scope and present the internal behavior of the Web service client, and discuss the method to bind Web services with Triple Space.

Since this deliverable focuses on the prototype, we also present the third-party dependencies and provide a usage guideline.

1.3 Target Audience

This document describes the main building blocks for the integration of Triple Space with WSMX. The target audience of this document includes researchers as well as practitioners that work in the areas of Triple Space (or tuple space), Web service, Semantic Web services, and Web service registry/discovery/invocation. Systems analysts and systems architects needing a thorough knowledge of Triple Space grounding to Web service technologies may also benefit from this document. Although no specific pre-knowledge is required to follow this document, basic knowledge in RDF, SOAP, WSDL, Triple Space, and WSMX may allow better following the document and for gaining more benefits from it. The work should be of interest to anyone involved with Semantic Web Services and more generally also in Service Oriented Architecture.

2 Implementation Architecture and Components Description

This section presents how the components of WSMX can be integrated using Triple Space. The possible integration between external components, i.e WSMX clients and internal integration is covered in this section.

2.1 Web Service Client

The purpose of this component implementation is twofold. First it will enable the clients of the Semantic Web Services execution environment (i.e. WSMX) to access it through Triple Space. WSMX's user API (or WSMX entry points) is exposed as a Web service which is then used by SOAP clients to submit Goals; the entry points are described through a WSDL (Web Service Description Language) document. Our implementation will enable Triple Space clients to access WSMX and submit the Goals via Triple Space. Secondly, it will also enable the WSMX invoker to invoke the external/end-point Web Services over Triple Space.

2.1.1 Scope

Clients accessing WSMX via Triple Space

Application services or middleware systems are typical examples of Triple Space clients who communicate using Triple Space API. In this scenario, a client can either be a service provider or a service requester involving either a read operation or a write operation to Triple Space. Semantic Web Services execution environment (WSMX) itself as being a service provider (i.e. semantic service execution) and its client as being service requesters, are conceived as Triple Space clients. The objective is to enable WSMX clients to communicate with WSMX via Triple Space. In this scenario, users communicate with WSMX either to find the services they are interested in or to register description of services they are offering. Users invoke *achieveGoal*( ) operation of WSMX communication manager to find services that they are interested in. The description of offered services are registered with WSMX by invoking *publish*( ) operation of WSMX communication manager.

Triple Space API provides a simple set of *read*, *write* operations to allow interaction between interacting parties. In addition, operations for subscribing to a particular message type (represented as rdf graph pattern) is provided by Triple Space API. For each subscription, if the matching graph pattern is available, Triple Space notifies to all who subscribed for this particular graph pattern. These simple operation can be mapped to the operations provided by WSMX communication manager API. Having such mapping at hand, the communication scenario described above can be mediated through Triple Space thereby avoiding synchronicity and reducing communication overhead.

WSMX invoking Web Services via Triple Space

WSMX after performing the dynamic service discovery, selection, composition and mediation (as required), requires to invoke the end-point external Web Services. In this case, WSMX itself acts as a service requester (on behalf of its client) and the

end-point service acts as service provider, which it the end are also conceived as Triple Space clients. The objective here is to enable the communication of WSMX with the end-point Web Services via Triple Space.

The *WSMX Web Service invoker* is part of the WSMX communication manager component and is responsible for mapping an invocation of a WSMX-external Web Service provider to the transport protocol supported by the Web Service. In combination with the WSMX grounding component, which implements a bi-directional mapping between semantic data communication between WSMX component (in form of WSMO objects) and the message and data format supported by WSMX-external Web Service endpoints, it enables invocation facilitates invocation of WSMX-external Web Service endpoints. Triple Space grounding for the WSMX invoker will enable it to support the invocation of two types of end-point Web Services, i.e. Web Services with Triple Space extensions, and Web Services with current W3C standards (i.e. without Triple Space extensions).

2.1.2 Internal Behavior

Clients accessing WSMX via Triple Space

The communication between WSMX and Triple Space is enabled by mapping the WSMX Communication Manager (CM) API with Triple Space API. The Communication Manager API provides necessary entry points for the clients to communicate with it. In D4.2 we have provided the mapping of Triple Space API with the Communication Manager API.

Assuming that WSMX has already registered its endpoint descriptions at Triple Space together with expected message graph, following logical sequence narrates the overall communication process.

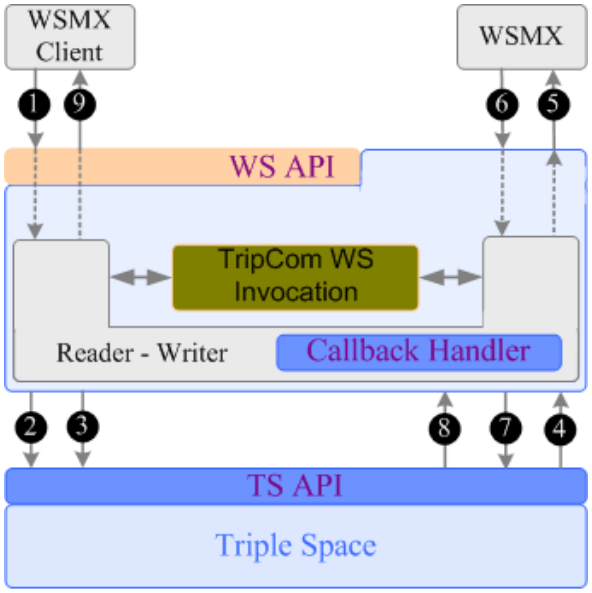

Figure 2.1: Internal behavior of WSMX Client invoking WSMX via Triple Space

1. If a WSMX user wants to provide a Goal to WSMX to execute, it submits the Goal to a triplespace via WS API, through a SOAP message.

2. The message is received by Triple Space WS invocation module, which takes care for the translation of the SOAP message into RDF triples, as well as the content of the message (i.e. Goal) to RDF.

3. The RDF triples are published on Triple Space by using TS API, in a sub-space where WSMX is already subscribed to receive the messages.

   It is to be noted that the WSMX is subscribed to event *request* and the WSMX clients are subscribed to *response*

4. As soon as the triples are published to a space where WSMX is subscribed to, an event triggers a notification to WSMX.

5. WSMX after receiving notification from Triple Space, retrieves the data from Triple Space through Triple Space invocation module.

6. WSMX processes and executes the Goal and publishes the response back to Triple Space Invocation Module.

7. The results from WSMX are transformed into appropriate RDF triples and are published to Triple Space in the space where the WSMX client is subscribed to.

8. As soon as the data is written to the event *response*, it generates notification for the WSMX client.

9. The WSMX client receives the data back from the WSMX.

WSMX invoking Web Services via Triple Space

The extended WSMX invoker with support for Triple Space transport protocol is dependent on an extended grounding component that implements the transformation of the invocation request from WSMO objects communicated inside WSMX to (i) a message format (e.g. in form of the SOAP-RDF representation proposed in D4.2) and (ii) message payload format that can be processed by the service provider.

1. WSMX invoker inside WSMX invokes the end-point/external Web Services which are exposed over Triple Space. As a first step, the WSMX invoker publishes the invocation request.

2. The message is received by Triple Space WS invocation module, which takes care for the representation of SOAP message into RDF.

3. The RDF triples are published on a triplespace using the TS-API in a sub-space where the end-point Web Service (enabled with Triple Space extensions) is subscribed to.

   It is to be noted that the end-point Web Services are subscribed to the event *request* and the WSMX invoker is subscribed to the event *response*.

4. As soon as the triples are published to a space where end-point Web Service is subscribed to, an event triggers a notification to the Web Service.

5. Web Service, upon notification, retrieves the message from Triple Space and process it.

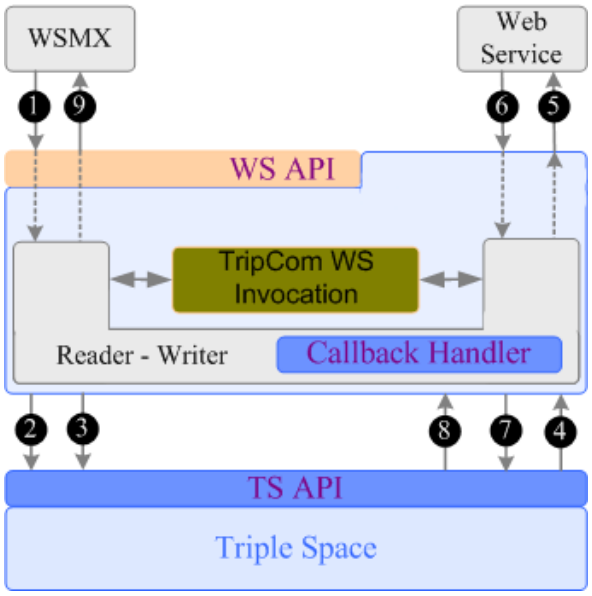

Figure 2.2: Internal behavior of WSMX Invoker invoking Web Service via Triple Space

6. The result from the Web Service is sent back from the Web Service to Triple Space WS invocation module, where it takes care for necessary conversions to represent in RDF.

7. The invocation response, from Web Service is then published in Triple Space where the WSMX invoker is subscribed to.

8. As soon as the data is published to the event *response*, it generates notification for the WSMX invoker.

9. WSMX invoker, upon notification, retrieves the result from Triple Space.

2.1.3 Limitations

The communication between WSMX client, WSMX and the end-point Web Services requires all the communication parties to be subscribed to appropriate events, for a successful communication.

2.2 Web Service Registry

To facilitate service discovery by service requesters, service providers need to publish descriptions of the services they provide to service registries. These descriptions comprise both *organizational* and *technical* descriptions. While organizational Web service descriptions are focused on categorizing and classifying services and providing information about the person or institution offering the service, technical Web service descriptions provide the information that is needed by service requesters to interact with a service provider, such as information about the location of the service, the service's supported communication protocols, supported operations and message types.

2.2.1 Scope

The Web service registry component aims at providing an implementation of a Triple Space-based (Semantic) Web service registry that facilitates storing and retrieving both technical and organizational Web service descriptions to and from Triple Space.

On the organizational level, the developed registry supports the UDDI [4] information model for describing and classifying both services and their corresponding service providers. For this purpose, the Web service registry as presented here integrates with the Web service registry prototype developed as part of *T4.3 "Implementation of Web service registry mechanisms in a Triple Space"*.

On the technical level the developed registry supports two types of technical Web service descriptions: Semantic Web service descriptions in WSML [7] and "regular" Web service descriptions in WSDL [3]. To link the technical descriptions of a Web service to its organizational descriptions, we follow the approach presented in [5] where for each technical description of a Web service (i.e. in case of the presented approach a WSDL file of the service) a corresponding tModel is created, which contains a reference to the service's technical description in the overviewURL property of the tModel's overviewDoc. This tModel is referenced from the service's businessService UDDI description through a tModelInstanceInfo reference.

In the context of Triple Space, the same approach can be used where the overviewURL points to the triplespace containing the technical Web service description in RDF representation, i.e. following the WSDL-RDF mapping provided as part of the WSDL 2.0 specification in case of Web services described in WSDL and following the WSML-RDF mapping [6] in case of services described in WSML. This way a client can retrieve the technical description of a Web service based on its organizational description.

Multiple approaches can be considered to facilitate discovery of Web services based directly on their technical service description.

Common super-space(s) In this approach, all triplespaces containing technical Web service descriptions are sub-spaces of one (or multiple) common registry super-space(s). Upon service discovery, service requesters can specify the identifier of the registry super-space (or multiple triplespace identifiers in case multiple spaces should be queried for discovery). Since the data retrieval operations of the TS API respect the triplespace hierarchy (i.e. the data set a query on a triplespace is evaluated on includes both the specified triplespace and in particular all its sub-spaces) a discovery operation succeeds if it can be fulfilled by either the specified triplespace or any of its subspaces.

Global discovery In scenarios in which the aforementioned approach is not feasible (e.g. due to clients not knowing about the respective registry super-space the Web service description is stored in), the global retrieval operations supported by the TS API (i.e. the consumption operations without a specified triplespace identifier) can be used to discover services in the global triplespace.

2.2.2 Internal Behavior

In Figure 2.3, the interaction of a user with the Web service registry component is depicted, along with the registry's internal components. The individual step in the interaction are outlined in the following.

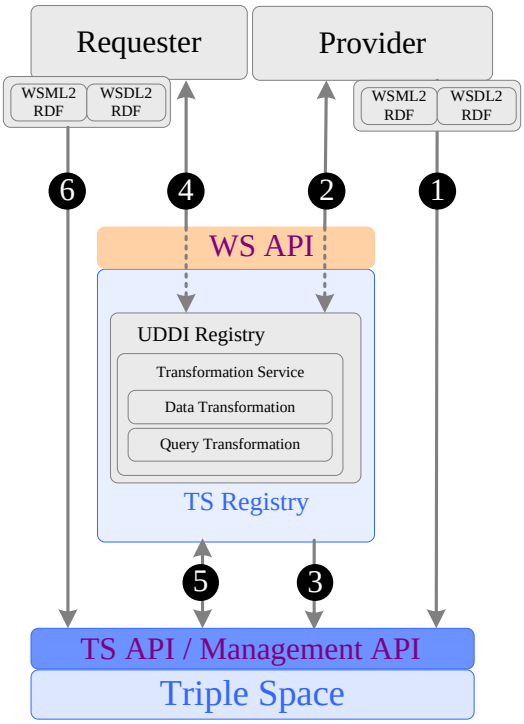

Figure 2.3: Internal behavior of Web service registry

1. The service provider registers the technical description of the Web service to be published. For this purpose, the service provider transforms the technical service description to RDF format using either *WSMO4RDF* in case the service is described through a WSML document or through the WSDL2RDF mapping prototype developed as part of D4.2. Prior to publication, the service provider might use the operations provided by the Management API to create a triplespace in which the service description is to be stored. Once transformed to RDF, the service description graph is stored using the operations of Triple Space API.

2. To publish organizational information about the published service, the service provider uses the UDDI interface of the registry (i.e. its saveService operation). To link the technical Web service description of the service (published in step 1) to its organizational description, the service provider references the triplespace URI of the space containing the technical service description as described in Section 2.2.1.

3. The registry maps the UDDI operations to Triple Space API operations; a description how this is done in detail is provided as part of D4.3.

4. To search/retrieve a service description from the registry, a service requester can query the registry using the registry's UDDI inquiry API (e.g. based on the service's classification and categorization information or its service provider). The result of this interaction is e.g. a UDDI serviceDetail message if the service requester queries for a businessService description.

5. The operations of the UDDI inquiry API are mapped to queries that are issued on Triple Space. The result of these queries is translated to instances of the UDDI information model and returned to the requester.

6. In case the service requester wants to retrieve the technical Web service description associated with the service's organizational description, the service provider can issue a corresponding query on the triplespace containing the service's technical description (stored in step 1). As described above, the URI of the triplespace the description is contained in is stored in the overviewURL of the overviewDoc of the businessService's tModelInstanceInfo element. Using the transformation service, the Web service description is transformed from its RDF representation back to its "native" format. The result of this interaction is the technical description of the service.

2.3 Web Service Discovery

This component aims to the implementation of a Web service discovery using Triple Space as the storage for WSML [6] Web services. The discovery request is represented as a WSML goal, which is used for matchmaking between Web services described using WSML. The discovery component is built upon WSMX discovery engine which supports different level of discovery ranging from keyword search to fully semantic search.

2.3.1 Scope

In Triple Space, the Web service discovery component aims to support two types of discovery [9, 10]:

- ***Keyword-based discovery***. The keyword-based discovery is a basic ingredient in a complete framework for semantic web service discovery. In a typical keyword-based scenario a keyword-based query engine is used to discover services. A query, which is basically a set of keywords, is provided as input to the query engine. The query engine match the keywords from to query against the keywords used to describe the service.
- ***Discovery based on semantic descriptions of services***. This type of discovery can be further divided into the following two categories: *Discovery based on simple semantic Descriptions of Services* and *Discovery based on rich semantic Descriptions of Services*. They use ontologies, which offer a formal and explicit specification of a shared conceptualization of some problem domain, as the controlled vocabularies with explicit and formal semantics, for supporting discovery [10].

2.3.2 Internal Behavior

The internal behaviour of the Web service discovery component is depicted in Figure 2.4, where the numbers specify different actions taken while executing a particular discovery request.

The actions as indicated by numbers in Figure 2.4 is the following.

1. A request for discovery, which is a WSML goal, is received from the user.

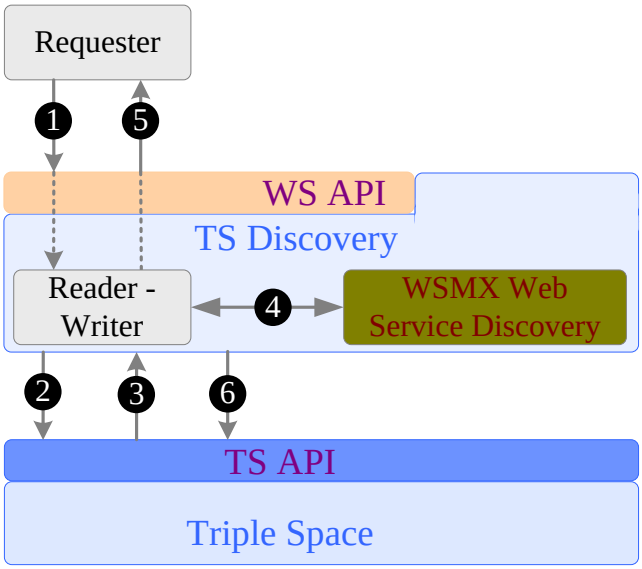

Figure 2.4: Internal behavior of Web service discovery

2. Triple Space Web service discovery component (TS Discovery) reads matching WSML Web services, which are registered in a triplespace, through TS API using "rd" operation.

3. WSML Web services retrieved from Triple Space are saved to the storage of WSMX, which is accessible to TS discovery component.

4. TS discovery component discovers and selects desired Web services for the given WSML goal using WSMX Web service discovery component.

5. The discovery result is handed over to the requester.

6. The event of this discovery, including the WSML goal and the result, is logged into Triple Space through TS API using "out" operation.

2.3.3 Limitation

The Web service discovery component assumes that the ontologies used by the goal and Web services are the same, and thus there is no ontological heterogeneity issues. This assumption introduces a limitation, i.e. the inability to resolve ontological heterogeneities. The reasons for this assumption, however, are based on the assumptions taken in WSMX discovery component which are: (i) WSMX discovery component does not use data mediation for solving possible data heterogeneity issues and assumes that the WSML goal and WSML Web services are using the same ontology, and (ii) mediation component in Triple Space is an optional component.

2.4 Web Service Invocation

The main purpose of Triple Space invocation component is to support asynchronous service interactions using Triple Space as the interaction platform, which is called

Triple Space-enabled Web service interactions as presented in Section 2.4.2, and specified in Section 4 in deliverable [12]. Unlike the traditional Web service interactions using HTTP where the involved Web services exchange their message in a direct and synchronous manner, Triple Space-based Web services exchange their messages (using TS API such as *rd*, *out* and *subscribe*) through Triple Space and thus achieve *time*, *location*, *reference* and *data schema* autonomy of Web service interactions [8].

2.4.1 Scope

The support of invocation is provided for two different kinds of Web services which are traditional Web services and the Web services with Triple Space enhancements [12]. In traditional Web services the descriptions are presented as WSDL messages where as in Triple Space enhanced Web services, the WSDL message also include Triple Space specific information. Details about how this information is represented is provided in D4.2 and D4.4.

- For traditional WSDL Web services, two kinds of invocation are supported: (1) direct service interactions which mean that Web services interact in a direct and synchronous manner, and (2) Triple Space-enabled Web service interactions.
- For Web services with Triple Space enhancements, Triple Space-enabled Web service interactions are supported through an implementation of the *Web service binding for Triple Space* as described in Section 2.4.4.

In addition, WDSL interaction patterns as specified in Section 3 in [12] are also supported over Triple Space.

2.4.2 Internal Behavior

The internal behaviour of the Web service invocation component for supporting Triple Space-enabled Web service interactions is depicted in Figure2.8, where the numbers specify different actions taken during the course of executing an invocation request.

The actions taken during the course of executing an invocation request as indicated by numbers in Figure 2.8 are the following.

1. A request for service invocation is received from the user (in their choice of format and protocol i.e., either legacy formats and protocols or those extended with Triple Space annotations as explained in D4.1 [11] and D4.2 [12]).

2. The invocation component is registered with the sub-space and subscribe two callbacks for two events: *InvocationRequest* and *InvocationResponse*.

   For a Web service with Triple Space enhancements, the sub-space to be registered is the sub-space prescribed in this Web service. We assume that Web services with Triple Space enhancements have been registered using Web Service registry component and thus the related spaces are created in Triple Space already.

   For a traditional Web service, an unique sub-space is needed to be created for registration purposes, and this sub-space needs to be dropped off when the invocation completes.

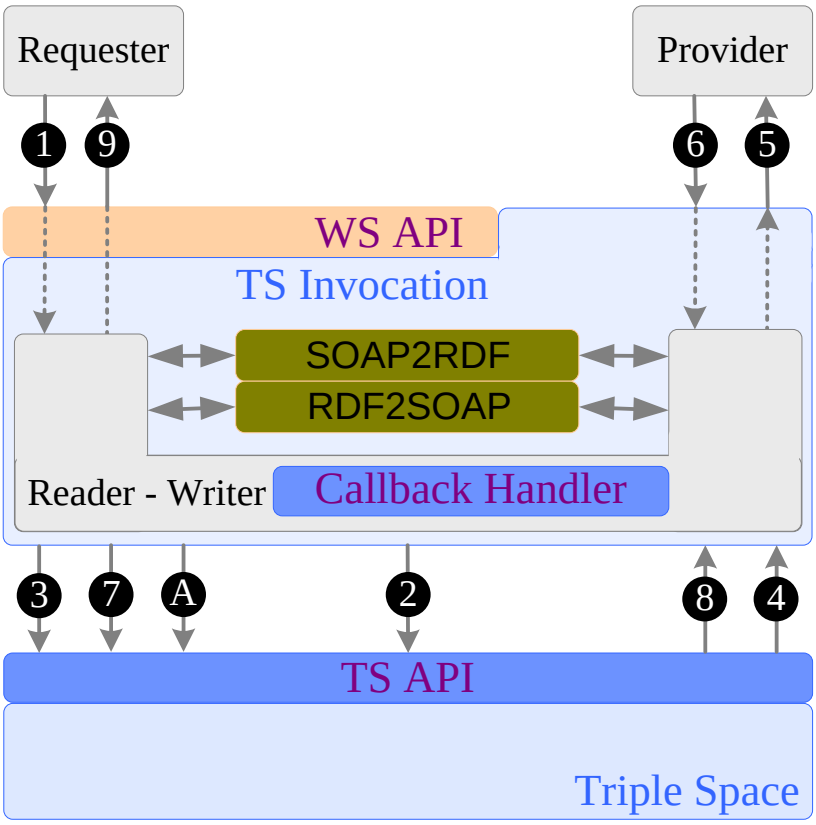

Figure 2.5: Internal behavior of Web service invocation

3. The received message is translated to RDF and sent to Triple Space using the TS API using "out" operation. This translation is achieved by means of SOAP-RDF mapping which is detailed in D4.2 [12].

4. As a result of action (3), Triple Space triggers the *InvocationRequest* event which is caught by the CallbackHandler, and the request is read from Triple Space through TS API using "destructive-read" operation.

5. The request read in RDF format is translated back to the format of the provider and actual Invocation is performed.

6. The result of the invocation is received.

7. This result is converted to RDF and written to Triple Space in the same way as in Step 3.

8. As in Step 3, Triple Space triggers the *InvocationResponse* event which is caught by the CallBackHandler and the response is read from the triplespace.

9. The result is converted back to the requesters format and protocol and handed over to the requester.

A. The event of this invocation, including the invocation request and the result, is logged into Triple Space through TS API using "out" operation.

2.4.3 Limitation

Besides providing different types of invocation support to both the traditional WSDL-based Web services and Triple Space-enabled Web services, there are few limitations of the proposed prototype of invocation component which are listed below.

- The invocation for WSML Web services is not supported because no WSML Web services are used in Triple Space. However, WSML Web service invocation can be supported by WSMX invocation component directly.
- A direct and synchronous service interaction is not supported for Web services with Triple Space enhancements.

2.4.4 Web Service Binding for Triple Space

Figure 2.6 depicts a schematic overview of the implementation of the Web Service Binding for Triple Space used by the WSMX Invoker component as described above.

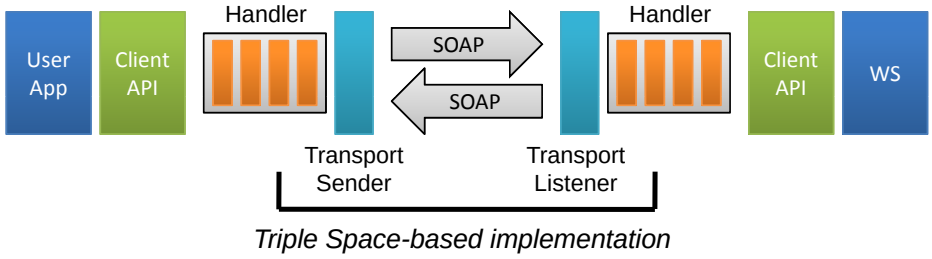

Figure 2.6: Schematic overview of Triple Space Web Service Binding prototype.

The prototype is realised as an extension of the open-source Web service runtime implementation *Apache Axis 2*[1] which provides a framework for processing SOAP envelopes and transmitting them between service requesters and providers. A client application interacts with Axis through a generic client API. Once a service requester performs an operation on the client API the operation is received by Axis and passed through a number of outbound handlers which provide implementations of various Web service standards such as WS-Addressing or WS-Security in form of a `MessageContext` object. When the `MessageContext` has passed the outbound handlers, it is received by the so-called *TransportSender*. The *TransportSender* is responsible (i) for encoding the request message in such a way, that it can be transmitted over the chosen (network) transport protocol and (ii) sending the message to the service provider. On the side of the service provider, the so-called *TransportListener* receives the incoming message an creates a corresponding `MessageContext` object containing the received message. The `MessageContext` is then passed through the inbound handlers which, among other things, dispatch the incoming message to the service implementation it is directed to.

Triple Space binding prototype is realised through Triple Space-specific implementations of the `TransportSender` and `TransportListener` interfaces which implement (i) the SOAP-RDF transformation and (ii) the message exchange patterns as explained in D4.2.

[1] `http://ws.apache.org/axis2`

2.5 Resource Manager

The purpose of the implementation of this component is to enable the Resource Management in WSMX using Triple Space. It will enable WSMX to persistently store the semantic descriptions (i.e. WSMO top level elements) in Triple Space. Secondly it will enable all the components of WSMX to communicate with each other by publishing and reading semantic data on Triple Space, rather than using direct synchronous interactions.

2.5.1 Scope

WSMX contains different repositories to store ontologies, goals, mediators and web services descriptions as WSML based files. The internal repositories of WSMX are required to be enabled to store WSML data as sets of RDF named graphs in Triple Space Storage. This is mainly concerned with transforming the existing representation of data in form of WSML into RDF representation. In addition the operations defined in the Resource Manager API have been mapped to TS API operations according to the mapping proposed in D4.2.

Secondly, all the WSMX components will be enabled to communicate with each other via Triple Space. D4.2 refers to interface WSMX manager and individual component wrappers with Triple Space Kernel in order to enable the WSMX manager to manage the components over Triple Space. The communication between manager and wrappers of the components will be carried out by publishing and subscribing the data as a set of RDF triples over Triple Space. All the data that is exchanged between the WSMX component are Web Service Modeling Language (WSML) based Goals and Web Service descriptions. In order to publish the WSML description on Triple Space, WSML description are serialized as a set of RDF triples to be stored as identifiable Named Graphs.

2.5.2 Internal Behavior

Considering the well-defined components in WSMX, the implementation approach should be component oriented and should only concern with Resource Manager. Triple Space grounding for Resource Manager should not be visible to other components. Resource Manager will provide the same standard interface for resource management to all other components in between, but it will be connected to Triple Space underline.

1. Any kind of data (i.e. Goals, Web Service descriptions, Mediators, Ontologies, event information) in WSMX is converted into a set of RDF triples.

2. This set of set of triples is published into Triple Space in a sub-space using TS API out operation.

3. The URI to access the sub-space (where the RDF triples have been published) is taken and sent back to WSMX Resource Manager.

4. The URI of the sub-space is saved by the WSMX Resource Manager for later use (i.e. when the data is required again).

5. The URI of the sub-space is used by the WSMX Resource Manager to retrieve the data from Triple Space.

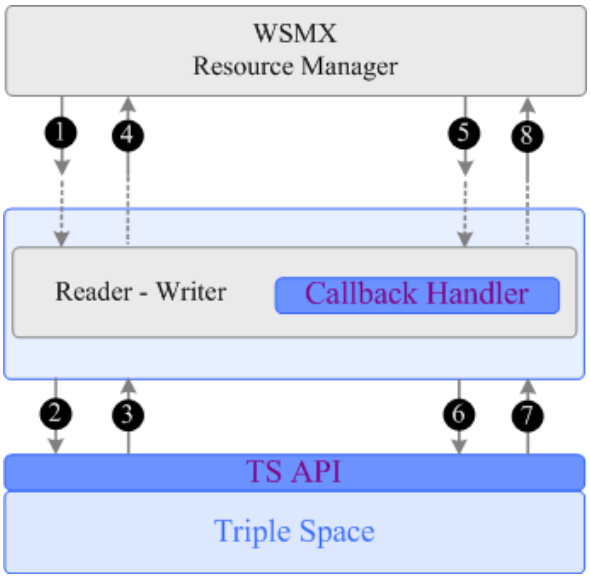

Figure 2.7: Resource Management in WSMX using Triple Space

6. The RDF triples are retrieved from Triple Space using TS-API operation rdWithSpace.

7 and 8. Retrieved triples from the sub-space are sent to the WSMX Resource Manager where it converts it back from RDF to appropriate WSML description.

Moreover, internal component communication in WSMX using Triple Space occurs when WSMX components communicate with each other by publishing and reading RDF triples in Triple Space, as shown in the figure.

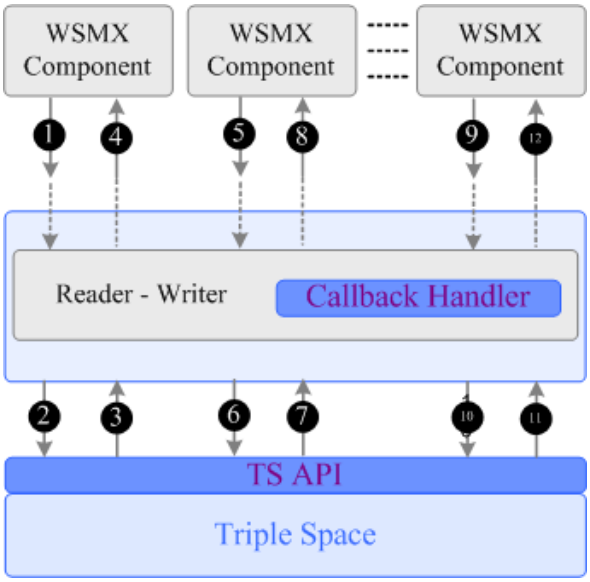

Figure 2.8: Component Management in WSMX using Triple Space

1, 5 and 9. Any component wishing to communicate with any other component writes the message.

2, 6 and 10. The message is transformed into appropriate RDF triples which are published in Triple Space using the TS-API for the target component.

It is to be noted that each of the WSMX components are subscribed to different events in order to be notified for the data intended for them. For example, Discovery Component is subscribed to event WSMXDiscovery, Selection Component is subscribed to event WSMXSelection, and similarly all the components of WSMX are subscribed to their respective events to be able to receive notification for their relevant data published in Triple Space.

3, 7 and 11. Whenever data is published for which a particular component is subscribed for, it receives a notification and the data is retrieved from the triplespace.

4, 8 and 12. Upon notification of publication of relevant data, the component retrieves the corresponding triples from Triple Space and converts them back into the format as required.

A general communication pattern for all the components of WSMX is described which is followed for internal communication and coordination within one WSMX system. It can include data communicated between WSMX manager and Discovery or Selection or Mediator or any other component.

2.5.3 Limitations

The current implementation plan is for the communication and coordination of components internal to WSMX only. It does not include communication between different WSMXs. While performing the implementation the approach was taken slightly different as described above, i.e. to keep it compliant with the current communication patterns of WSMX components over SOAP as well as JavaSpaces. Therefore, rather than replacing the communication completely, TripCom based communication was introduced as an additional means of communication where the components and their interfaces in WSMX are pre-known to each other and are deployed on same machine.

The event data is not changed or moved directly but is managed as, a simple RDF graph is modeled to be communicated between the components. The event logic is represented in Triple Space based on three kinds of information, i.e. (1) Name of Component, (2) Method to be invoked and (3) Arguments or output to the method. Two sub-spaces for each of the component in WSMX (one of input and one for output) are introduced for this purpose. One requiring to communicate with the component publishes input in the input subspace of the component (i.e. resourcemanagement input subspace) and the output is then published by the component in its output subspace (i.e. i.e. resourcemanagement output subspace). In this way we do not need to translate all the event and intermediate memory data in to RDF and back and to avoid delays in unnecessary transformation. For the demonstration purpose, the communication of Resource Manager will be shown that, for example if it is asked for loading SWS descriptions from Triple Space, a triple is published in the Resource Manager input subspace, and the Resource Manger after loading the SWS description from Triple Space publishes answer in the Resource Manager output subspace.

As a future work, we envision to have multiple distributed instances of WSMX, deployed at different locations, required to coordinate with each other. In that case the total number of WSMX instances and their components will not be pre-known and

usefulness of TripCom based communication could be further exploited where communication parties do not have to know each other and are supposed to be maximally decoupled.

3 Third party dependency

This section lists the third party dependencies.

3.1 JAVA JDK 1.5

LICENSE
The Sun license for the Java SDK 1.5.0+ is available at:
http://java.sun.com/j2se/1.5.0/jdk-1_5_0_16-license.txt
Third party licenses available at:
http://java.sun.com/j2se/1.5.0/j2se-1_5_0-thirdpartyreadme.txt
OBTAIN A LICENSE
The license is granted by accepting the proposed conditions at download time.
DOWNLOAD
The Windows installer is available for download at the url:
http://java.sun.com/javase/downloads/index_jdk5.jsp
INSTALLATION
Simply run the installer.
NOTE: Java 5 has to be installed because Blitz does not work with Java 6

3.2 JAVA JDK 1.6

LICENSE
The Sun license for the Java SDK 1.6.0+ is available at:
http://java.sun.com/javase/6/jdk-6u10-license.txt
Third party licenses available at:
http://java.sun.com/javase/6/javase-6-thirdpartyreadme.txt
OBTAIN A LICENSE
The license is granted by accepting the proposed conditions at download time.
DOWNLOAD
The Windows installer is available for download at the url:
http://java.sun.com/javase/downloads/index.jsp
INSTALLATION
Simply run the installer.

3.3 Apache Axis 2

LICENSE
Axis2 is under Apache License, Version 2.0.
See http://www.apache.org/licenses/LICENSE-2.0.html
DOWNLOAD
http://ws.apache.org/axis2/download.cgi

3.4 Sesame 2.x

LICENSE
Sesame 2.x is under a BSD-style license. See http://www.openrdf.org/license.jsp
DOWNLOAD
http://www.openrdf.org/download.jsp
NOTE: Used for invocation module.

3.5 WSMO4RDF

LICENSE
WSMO4RDF is under Lesser General Public License (LGPL).
See https://ordi.svn.sourceforge.net/svnroot/ordi/trunk/wsmo4rdf/LICENSE
DOWNLOAD
http://ordi.sourceforge.net/source-repository.html

3.6 WSMX 0.5

LICENSE
WSMX is under GNU Library or Lesser General Public License (LGPL). See http://sourceforge.net/projects/wsmx/
DOWNLOAD
http://sourceforge.net/project/platformdownload.php?group_id=113321

3.7 jUDDI

LICENSE
jUDDI is under The Apache Software License, Version 2.0.
See http://ws.apache.org/juddi/license.html
DOWNLOAD
http://ws.apache.org/juddi/releases.html

4 User guideline

This section presents how to download and use the prototype.

4.1 Installation

4.1.1 WSMX Installation

In this section we explain a step by step WSMX installation on a Windows operating system. Get the WSMX from Triple Space SVN with updates on Triple Space integration.

Following software is necessary to run WSMX:

J2SE SDK

- Step 1. Download and install Java(TM) 2 SDK, Standard Edition 1.5.0 from `http://java.sun.com`.
- Step2. Setup the environment variable JAVA_HOME and point it to where the java has been installed (say `c:\java`).

WSMX configuration

- Step 1. Go to the WSMX directory in Triple Space SVN and copy it to your hard drive (say `c:\wsmx`)
- Step 2. Go to `PATH\wsmx` (where you have placed the extracted files)
- Step 3. Run the start.bat to run WSMX
- Step 4. Open internet browser and type URL `http://localhost:8081` If the WSMX server page is visible, WSMX has been installed and running successfully.

4.1.2 Triple Space kernel Installation

Compile and install the TipCom kernel as described in Readme.txt file available along with its distribution at Triple Space SVN.

5 Conclusions

Leveraging the works presented in previous deliveries in WP4, this prototype work describes the integration of Triple Space Computing with WSMX and shows how Triple Space can be used as a Semantic Web Services platform. We have presented our implementation work from the following two different but complementary perspectives:

- *The internal integration* which presents how WSMX's components can be integrated using Triple Space. This perspective includes (i) Web service registry component which aims at providing support for publishing descriptions of the services they provide to service registries, (ii) Web service discovery component which aims at providing support for discovering WSML Web services for a given WSML goal where WSML Web services are stored in Triple Space, and (iii) resource management which aims at enabling WSMX to persistently store the semantic descriptions (i.e. WSMO top level elements) in Triple Space. Secondly it will enable all the components of WSMX to communicate with each other by publishing and reading semantic data on Triple Space, rather than using direct synchronous interactions.

- *The external integration* which presents how the client can interact with WSMX through Triple Space. This perspective provides the following functionalities accessible to the clients: (i) it enables the clients of Semantic Web Services execution environment to access it through Triple Space. User API (or WSMX entry points) of WSMX is exposed as a WSDL description which is then used by SOAP clients to submit a Goal. The prototype implemented as part of the integration tasks will enable Triple Space clients to access the WSMX and submit the Goals via Triple Space, (ii) it enables the WSMX invoker to invoke the external Web Service end-points over Triple Space through Web service invocation component, which aims at supporting asynchronous service interactions using Triple Space as the interaction platform, which is called *Triple Space-enabled Web service interactions* as specified in Section 4 in [12].

In addition, third-party dependencies, i.e. information on softwares that the prototype uses and their licensing information is also provided. This deliverable finally provided a usage guide which tells how to install, configure and execute the developed tools and prototypes.

References

[1] Wsmo4rdf. Available at: http://www.ontotext.com/ordi/ORDI_SG/wsmo4rdf.html.

[2] D. Fensel J. Miguel Gomez A. Haller T. Haselwanter M. Kerrigan A. Mocan M. Moran E. Oren B. Sapkota I. Toma J. Viskova T. Vitvar M. Zaremba M. Zaremba C. Bussler, E. Cimpian. Web Service Execution Environment (WSMX). In *W3C Member Submission, June 2005*. Available at http://www.w3.org/Submission/WSMX, 2005.

[3] R. Chinnici, M. Gudgin, J. J Moreau, J. Schlimmer, and S. Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. *W3C Working Draft*, 26, 2004.

[4] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers et al. Uddi version 3.0.2. Oasis standard, UDDI Consortium, February 2005.

[5] F. Curbera, D. Ehnebuske, and D. Rogers. Using WSDL in a UDDI Registry, Version 1.07. `http://www.uddi.org/pubs/wsdlbestpractices.pdf`.

[6] J. de Bruijn, J. Kopeck, and R. Krummenacher. Rdf representation of wsml, d32 wsml working draft, 2006. Available at: `http://www.w3.org/Submission/WSMX`.

[7] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel. D16. 1v0. 2 The Web Service Modeling Language WSML. *WSML Final Draft March*, 20, 2005.

[8] C. Bussler et al. Web service execution environment (wsmx), w3c member submission, 2005. Available at: `http://www.w3.org/Submission/WSMX`.

[9] Uwe Keller, Ruben Lara, Holger Lausen, Axel Polleres, Livia Predoiu, and Ioan Toma. Wsmo discovery engine. WSML Working Draft. Available at http://www.wsmo.org/, November 2004.

[10] Uwe Keller, Ruben Lara, Axel Polleres, Ioan Toma, Michael Kifer, and Dieter Fensel. Wsmo web service discovery. WSML Working Draft. Available at http://www.wsmo.org/, November 2004.

[11] Brahmananda Sapkota, doug foxvog, Daniel Wutke, Daniel Martin, Martin Murth, Omair Shafiq, Andrea Turati, Emanuele Della Valle, Nuria Sanchez, and Jacek Kopecky. Architectural Integration of Triple Spaces with Web Service Infrastructures. Technical report, TripCom, FP6 - 027324, 2007.

[12] Omair Shafiq, Dario Cerizza, Jacek Kopecky, Daniel Martin, Martin Murth, Brahmananda Sapkota, German Toro del Valle, Andrea Turati, and Daniel Wutke. TripCom Grounding for Semantic Web Services. Technical report, TripCom, FP6 - 027324, 2007.