



D21.v0.1 WSMX Triple-Space Computing

WSMO Working Draft 02 February 2005

This version:

<http://www.wsmo.org/2005/d21/v0.1/20050202>

Latest version:

<http://www.wsmo.org/2005/d21/v0.1/>

Previous version:

<http://www.wsmo.org/2004/d21/v0.1/20041208>

Authors:

Brahmananda Sapkota
Edward Kilgarrif
Francisco Jose Martin-Recuerda Moyano
Ioan Toma
Reto Krummenacher

Editors:

Brahmananda Sapkota
Francisco Jose Martin-Recuerda Moyano

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

Copyright © 2005 [DERI](#), All Rights Reserved. [DERI](#) liability, trademark, document use, and software licensing rules apply.

Executive Summary

This deliverable will identify and describe the concept of Triple Space Computing [[Fensel D., 2004](#)] (TSC) and find its scope in WSMX [[Oren et. al., 2004](#)]. It will specifically concentrate on facilitating asynchronous communication between geographically distributed WSMXs providing a shared RDF space between them. To identify the scope of TSC in WSMX, a brief study of WSMX will be done and relationship between WSMX and TSC will be presented. Finally, interaction between WSMXs through TSC has been presented.

Table of contents

1. [Introduction](#)
 - 1.1 [Purpose of this Document](#)
 - 1.2 [Document Overview](#)
 2. [Web Service Execution Environment \(WSMX\) Overview](#)
 3. [Tuple Space Computing](#)
 4. [Triple Space Computing](#)
 5. [Publish Subscribe Paradigm](#)
 6. [Integration of Triple-Space Computing Architecture with WSMX](#)
 7. [Interaction Between WSMXs Through TSC](#)
 8. [Conclusions and Future Work](#)
- [References](#)
- [Acknowledgment](#)
-

1. Introduction

Triple-Space Computing is a simple and powerful paradigm that inherits the communication model from Tuple Space Computing (TSC) model and projects in the context of Semantic Web Service [Fensel, 2004]. It is based on the evolution and integration of other known technologies such as Tuple-Space Computing [Gerlernter, 1992], Persistence Message-based Architecture [Alonso et. al., 1995], Semantic Web and on RDF Schema [Brickley and Guha, 2004]. It provides a persistent shared space for participating applications to enable seamless interaction without having the need of direct message exchange between them. Web Service Execution Environment (WSMX) enables the dynamic discovery, selection, mediation, invocation, and interoperation of Semantic Web Services [Oren et. al., 2004]. WSMX in its current state supports synchronous communication with other WSMXs and or systems. While integration process incorporates large number of systems, more than one WSMX will be engaged for facilitating such integration processes. In addition, one WSMX may require to communicate with other WSMX in the due course. Such communication will become costly when WSMXs are heavily loaded and the processing time is high. In such cases TSC provides a medium for asynchronous communication minimizing the cost of interaction between WSMXs. One of the main benefits of asynchronous communication is that, unlike in synchronous communication, participants can come and go without hindering the communication process.

1.1 Purpose of this Document

This document describes the possibility of using TSC in WSMX to enable distributed asynchronous communication between WSMXs. This document starts with the definition of TSC and WSMX; finding the possible relationships between them. One of the main focuses of WSMX-TSC is to provide a shared space for WSMX and provide new means of asynchronous communication.

1.2 Document Overview

An overview of WSMX is presented in section two. As part of the background information, section three introduces tuple space computing. Triple Space Computing is presented in section three of the document. Likewise, publish-subscribe paradigm is presented in section five and section six highlights possible integration of TSC and WSMX architecture to enable inter WSMX interaction. Section seven is describes asynchronous communication between WSMXs through TSC and finally, section five concludes the document and looks at future works.

2. Web Service Execution Environment (WSMX) Overview

The Web Services Execution Environment (WSMX) [Zaremba et al., 2004] is an execution environment for dynamic discovery, selection, mediation and invocation of semantic web services. WSMX builds on the Web Services Modelling Ontology (WSMO) [Roman et al., 2004] that describes various aspects related to semantic web services.

WSMO is based on four concepts: web services, ontologies, goals and mediators. Web services are units of functionality; every web service has exactly one capability, which describes logically what this web service can offer. Every web service has a number of interfaces, which specify how to communicate with it. Goals describe some state that a user may want to achieve. Ontologies are the formal specification of the knowledge domain used by both the web service to express its capability, and by the goal to express the desired world state. Mediators are used to solve different interoperability problems, like differences in ontologies used by a web service and a goal. WSMX is developed as a reference implementation of an execution environment for web services. WSMX manages a repository of web services, ontologies 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 Architecture:

In this section we use the term architecture to introduce the abstract software components that make up WSMX. The WSMX Manager and the Execution Engine co-ordinate the activities of WSMX following the execution semantics defined in [Oren, 2004]. All data handled inside WSMX is represented internally as an event with a type and state. The WSMX manager manages the processing of all events passing them to other components in the logic layer as appropriate.

The MatchMaker is responsible for matching goals to web service capabilities. In the event that multiple web services are found matching a specific goal, the Selector is invoked to select the web service that best fits the requirements of the goal's owner. The Mediator component provides a means of transforming data based on concepts in one ontology to data based on concepts in another ontology. The mapping is based on rules defined between concepts in the source and target ontologies.

In the event that data mediation is required and the mediated data is in a non-XML format, the XML Converter can be invoked to translate the results of the Mediator into XML. This is necessary as the web service invocations are via SOAP and the message format for SOAP messages is XML. The Compiler component parses WSML messages received from the WSMO Editor in the User Interface layer, validates the messages against WSMO and then stores the message elements in WSMX repository. The elements compiled to WSMX are the metadata for web services, ontologies, mediators. Once any of these elements have been compiled to WSMX, they are available for use during execution of goals sent to WSMX.

The Message Parser parses the WSML Messages containing goals sent to WSMX. The goal is parsed and stored persistently. The functionality of the Message Parser is similar to that of the compiler but there is a conceptual difference. The Message Parser operates on instances of goals while the Compiler operates on the metadata for web services, ontologies, and mediators. Adapters allow applications which can not directly communicate with the interfaces provided by WSMX to communicate with WSMX. The Invoker is responsible for making invocations to web services as part of the execution of a goal. Invocations are based on the WSML description of the web service. The Repositories are used to store the definitions of goals, web services, ontologies and mediators within WSMX. The repositories can be either internal to WSMX or external as, for example, the API to the UDDI described in the WSMO Registry [Herzog et al., 2004]. The figure below shows the WSMX architecture and the position of each component with it.

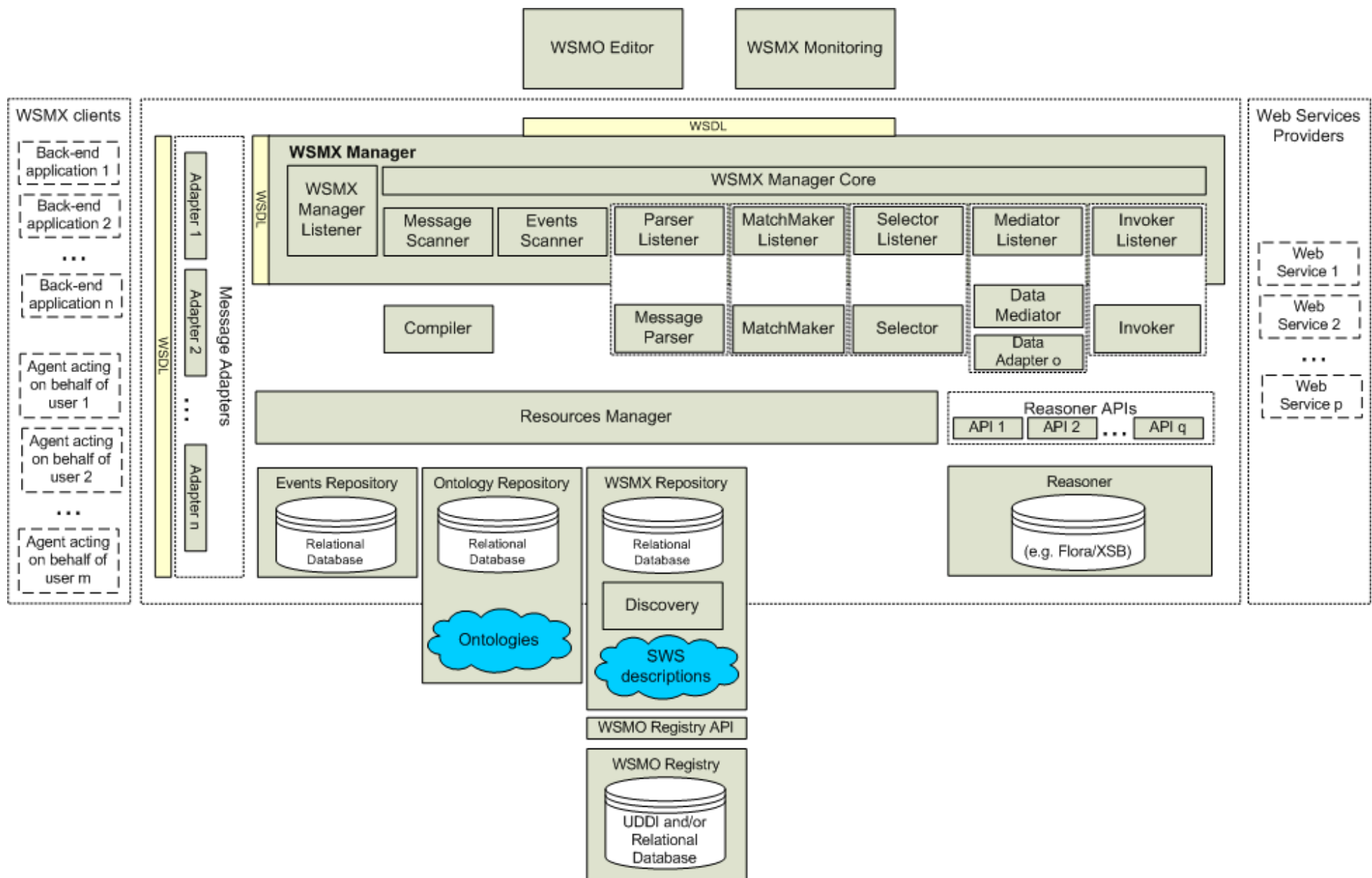


Figure 1: WSMX-Architecture

Messages & Workflow:

WSMX is actually based on the concept model of an event architecture. Components subscribe to a core component, WSMX Manager. This manager generates events for the subscribed components. The listener of each of these components can create, update and consume events. The interaction between components is done by events that contain messages. These events are java objects with an internal data structure and an interface specification. The execution semantics of WSMX as described in [Oren et al., 2004] is implemented in this component. Consequently the WSMX Manager

is responsible for coordinating the overall operation of a WSMX system.

The event listener components are capable of accepting and processing events picked up by the events scanner from the event repository (in other words: the events scanner queries the event repository looking for "unlocked" events and if it finds any, the broadcasting mechanism of the WSMX manager core distributes this information to each listener of each WSMX component). The event repository is the persistent mechanism where all the events that are processing by the system are stored.

A design based in Service Oriented Architecture and a workflow engine that will allow the definition of multiple execution semantics are the main goal for future revisions of WSMX architecture. This improved architecture will allow for easy to plug and unplug components and to integrate components from different vendors/developers.

3. Tuple Space Computing

To be written

4. Triple-Space Computing (TSC)

Triple Space Computing (TSC) [Fensel, 2004], follows the same goals for Semantic Web Services than the Web achieved for humans: re-define and expand previous ways of communication (cf. Figure 2). TSC will become in the necessary infrastructure where Semantic Web and Semantic Web Services will become true, and as [Fensel, 2004] pointed out: "Triple Space may become the web for machines as the web based on HTML became the Web for humans".

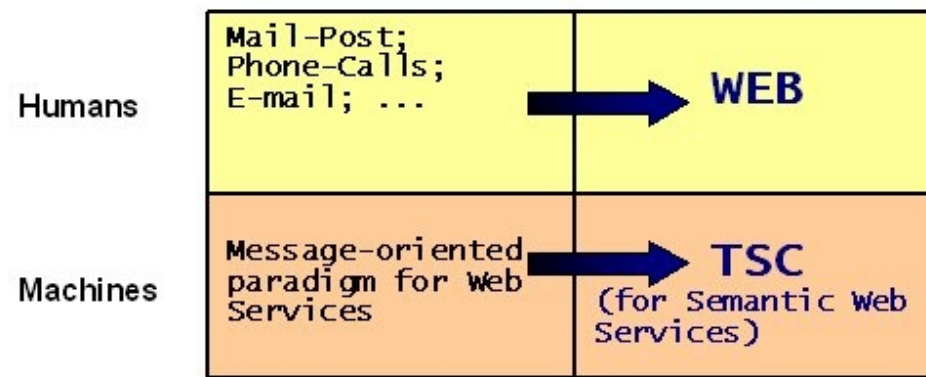


Figure 2: Evolution of communication mechanisms for humans and machines

TSC is based in the evolution and integration of several well-known technologies: Tuple Space Computing [Gerlinter, 1992], Shared Object Space (<http://www.tecco.at/en/eProducts.html>), Persistent Message-based Architecture [Alonso et. al., 1995], Semantic Web and in particular RDF Schema [Brickley and Guha, 2004]. TSC is based on the communication model of Tuple Space Computing: Instead of sending messages forward and backward much simpler means for communication are provided. Processes can write, delete, and read tuples from a global persistent space.

- Tuple or space-based computing has one very strong advantage: It de-couples three orthogonal dimensions involved in information exchange (cf.

Figure 3): reference, time, and space.

- Processes communicating with each other do not need to know explicitly from each other. They exchange information by writing and reading tuples from the tuplespace, however, they do not need to set up an explicit connection channel, i.e., reference-wise the processes are completely de-coupled.
- Communication can be completely asynchronous since the tuplespace guarantees persistent storage of data, i.e., time-wise the processes are completely de-coupled.
- The processes can run in completely different computational environments as long as both can make access to the same tuplespace, i.e., space-wise the processes are completely de-coupled.

This strong decoupling in all three relevant dimensions has obvious design advantages for defining reusable, distributed, heterogeneous, and quickly changing applications like promised by web service technology. Also, complex APIs of current web service technology boil down to a read and write operation in a tuplespace. Notice that a service paradigm based on the tuple paradigm also revisits the web paradigm: information is persistently written on a global place where other processes can smoothly access it without starting a cascade of message exchanges.

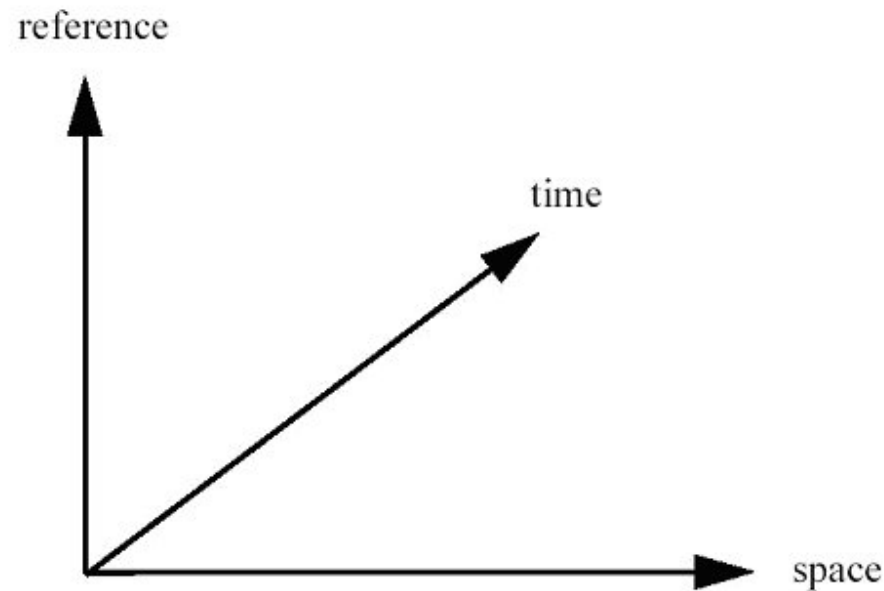


Figure 3: Three separate dimensions of cooperation [Angerer, 2002]

However, [Johanson and Fox, 2004] reports some shortcomings of the current tuple space models. They lack any means of name spaces, semantics, unique identifiers and structure in describing the information content of the tuples. This tuple space provides a flat and simple data model that does not provide nesting, therefore, tuples with the same number of fields and field order but different semantics cannot be distinguished.

We propose a simple and promising solution for this problem extending the tuple space into a triple space, where <subject, predicate, object> triples describe content and semantics of information. The object can become a subject in a new triple and so defining a graph structure capturing structural information. Fortunately, with RDF [Klyne and Carroll, 2004] this space already exists on the web and provides a natural link from the space-based computing paradigm into the semantic web. Notice that the semantic web is also not becoming unnecessary based on the Tuple-spaced paradigm, i.e. we envision current Semantic Web technologies and Tuple Space as the building blocks for a "Global Semantic Space" on the web. The global space can

help overcome heterogeneity in communication and cooperation: The Tuple Space simply brought to a global scale only does not provide any answer to data and information heterogeneity. Nevertheless, this aspect is what the semantic web is all about, and what we aim to fruitfully combine.

To make this vision real, two core components are currently identified. The first one is the technical infrastructure that it will include a distributed persistent mechanism for storing triples and should also offer the following desirable features: asynchronous communication, reliability, high-availability, robustness and security. The second, core component is the ontological infrastructure that will provide a semantic specification of the domain where business processes will interact each other.

5. Publish Subscribe Paradigm

To be written

6. Integration of TSC Architecture with WSMX

To be written

7. Interaction Between WSMXs Through TSC

In its current state, WSMX supports synchronous communication with other WSMXs. Synchronous communication is necessary when immediate responses are required. However, when there is high response latency because of some reason, e.g., network congestion, slow processing, third party invocation, etc, the synchronous communication will be costly as it forces the system (component) to remain idle for a long time. As WSMX may invoke other WSMXs and Web Services, it may not be able to provide immediate responses. Such problems can be addressed by using TSC as communication channel between WSMXs (Cf. [Figure 4](#)), as TSC supports purely asynchronous communication. Enabling asynchronous communication between WSMXs brings WSMX as step close to achieving its architectural goal i.e., to support greater modularization, flexibility and decoupling between communicating WSMXs. Similarly, it enables WSMX to be highly distributed and easily accessible.

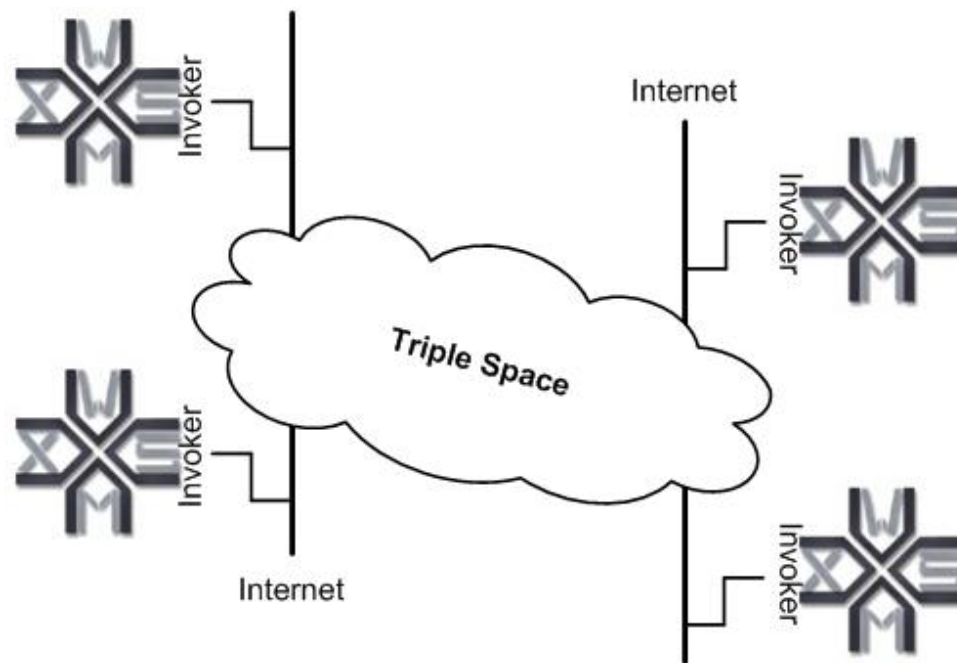


Figure 4: Communication between WSMXs through TSC

The communication between WSMXs occurs in the following manner. A sending WSMX writes its message (that need to be sent to the receiving WSMX) onto the Triple Space (TS). While writing, the sending WSMX also includes its own address, the address of the receiving WSMX and the message that needs to be delivered to the receiving WSMX. This means that the 'write' operation will look like the following: **write** <source, destination, data>. TS notifier will then notify the receiving WSMX, when it is available, about the arrival of the data destined for it. When the receiving WSMX requires this data, it will read the data from the TS. The 'read' operation will look like the following: **read** <destination>. Where *destination* is the destination address specified by the sending WSMX while doing the write operation. In order to guarantee the transactability, when the receiving WSMX reads the data, TS notifier will notify the sending WSMX that the data has been consumed by the receiving WSMX. This data when no longer needed will then be removed from the TS.

8. Conclusions and Future Work

In its current state, this document presents only some sections of what we intend to do in this deliverable. Future work will provide elaborated specification of the TSC functionalities that will be used to support asynchronous communication between WSMXs.

A. References

[Alonso et. al., 1995], G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, R. Günthör, M. Kamath: Exotica/FMQM: A Persistent Message-Based Architecture, In the proceedings of IFIP Working Conference on Info Sys for Decentralized Organizations, Trondheim, August 1995; Also available as IBM Research Report RJ9912, IBM Almaden Research Center, November 1994.

[Angerer, 2002], B. Angerer: Space Based Computing: J2EE bekommt Konkurrenz aus dem eigenen Lager, Datacom, no 4, 2002.

[Brickley and Guha, 2004], D. Brickley and R.V. Guha (eds.): RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, February 2004, <http://www.w3c.org/TR/rdf-schema/>

[Fensel D., 2004], D. Fensel: Triple-based Computing. DERI Research Report, 2004-05-31.

[Gerlernter, 1992], D. Gerlernter: Mirrorworlds, Oxford University Press, 1992.

[Herzog et al., 2004], R. Herzog, P. Zugmann, M. Stollberg, and D. Roman(ed.): WSMO Registry. WSMO Working Draft D10, <http://www.wsmo.org/2004/d10/v0.1/>

[Johanson and Fox, 2004], B. Johanson and A. Fox: Extending Tuplespaces for Coordination in Interactive Workspaces, Journal of Systems and Software, 69(3), January 2004:243-266.

[Klyne and Carroll, 2004], G. Klyne and J. J. Carroll (eds.): Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, February 2004, <http://www.w3.org/TR/rdf-concepts/>

[Oren, 2004], E. Oren. WSMX Execution Semantics. WSMO Working Draft v0.1, <http://www.wsmo.org/2005/d13/d13.2/v0.2/>

[Oren et. al., 2004], Moran M. and Zaremba M.: Overview and Scope of WSMX, WSMO Working Draft v0.1, <http://www.wsmo.org/2004/d13/d13.0/v0.1/>

[Roman et al., 2004], D. Roman, H. Lausen, and U. Keller. Web Services Modeling Ontology Standard. WSMO Working Draft v02, <http://www.wsmo.org/2004/d2/v0.2/>

[Zaremba et al, 2004], M. Zaremba, A. Haller, Maciej Zaremba, and M. Moran: WSMX - Infrastructure for Execution of Semantic Web Services, 2004

Acknowledgment

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

The authors would like to thank to all the [members of the WSMO working group](#) for their advises and inputs to this document.

