D 2.4.8.1 v1: Technical and ontological infrastructure for Triple Space Computing

Coordinator: Francisco Martín-Recuerda (UIBK)

Lyndon J.B. Nixon (FU Berlin), Elena Paslaru Bontas (FU Berlin) and James Scicluna (UIBK)

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB
Deliverable D2.4.8.1 v1 (WP2.4)
Semantic Web Services have inherited the Web Service communication model, which is based on synchronous message exchange and is not at all Web-like, as the Web is based on the model of persistent publish and read. Tuplespace-based communication offers the potential to remodel Semantic Web Service communication in a way that is more Web-like, bringing with it advantages of concurrency, asynchrony and co-ordination.

Document Identifier:	KWEB/2005/D2.4.8.1/v1
Class Deliverable:	KWEB EU-IST-2004-507482
Version:	V1
Date:	February 14, 2006
State:	Final
Distribution:	Public

Knowledge Web Consortium

This document is part of a research project funded by the IST Program of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) – Coordinator
Institute of Computer Science,
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polythechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne.
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse, 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe - Montbonnot
Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi – Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom.
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB
Universität Karlsruhe

Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer Science, University of Manchester,
Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed writing parts of this document:

University of Innsbruck (UIBK)
Freie Universität Berlin (FU Berlin)

Changes

Version	Date	Author	Changes
0.1	15-10-2005	Francisco Martin-Recuerda	Structure and initial content
0.2	18-11-2005	Lyndon Nixon	Added 3.3 and some comments
0.3	28-11-2005	Lyndon Nixon	Added 3.1 and more detail to Semantic Web Spaces (in ch 3, 4 and 5)
0.4	29-11-2005	Francisco Martin-Recuerda	Added section 3.2 and 3.4
0.5	06-12-2005	Lyndon Nixon	Expanding on Semantic Web Spaces in chapter 4
0.6	10-12-2005	Francisco Martin-Recuerda	Revision of section 3.4 and 4. New Structure.
0.7	21-12-2005	Elena Paslaru Bontas	Revision of the complete deliverable, including references; expanding minimal architecture
0.8	02-01-2006	Elena Paslaru Bontas	Changed chapter 1.
0.9	03-01-2006	Elena Paslaru Bontas	Revised section 2.3.2
0.91	12-01-2006	Francisco Martin-Recuerda	Revised sections 1, 2.2, 2.4, 3, 4
0.93	20-01-2006	Francisco Martin-Recuerda	Revised all sections
0.94	30-01-2006	Lyndon Nixon	Proof read deliverable
0.95	06-02-2006	Lyndon Nixon	Slight revision following QA
0.97	09-02-2006	Francisco Martin-Recuerda Lyndon Nixon	Slight revisions following QA
1.0	14-02-2006	Francisco Martin-Recuerda	Final version

Executive Summary

Semantic Web Services have inherited the Web Service communication model, which is based on synchronous message exchange and is not at all Web-like, as the Web is based on the model of persistent publish and read. Space-based communication offers the potential to remodel Semantic Web Service communication in a way that is more Web-like, bringing with it advantages of concurrency, asynchrony and co-ordination.

In this deliverable, we consider four currently emerging proposals for space-based communication in the Semantic Web. Based on our analysis, we determine a prototypical model for persistent space-based computing in a Semantic Web Service environment.

Contents

1 Motivation for tuplespace-based computing

Web Services based on the message-exchange paradigm are not fully compliant with core paradigms of the Web itself. Instead of publishing the information based on a global and persistent URI, Web services establish stateful conversations based on the hidden content of messages. Besides being in contradiction with the basic design principles of the Web and the REST architecture [Fielding, 2000], the negative effect of such distributed applications that communicate via message exchange is that they require a strong coupling in terms of reference and time. This means that traditional Web Services require that the sender and receiver of data:

(1) maintain a connection at the very same time
(2) know each other, and
(3) share a common data representation.

The communication has to be directed to a particular service, and it is synchronous as long as neither party implements asynchronous communication (and jointly agrees on the specific way this mechanism is implemented).

We illustrate the aforementioned issues in terms of an eTourism use case [Stollberg et al., 2004], in which an employee of DERI Innsbruck, called James, wants to book a train and a hotel for the Knowledge Web plenary meeting at Trento. The start-up company VTA provides tourism and travel services based on Semantic Web technology (figure 1).

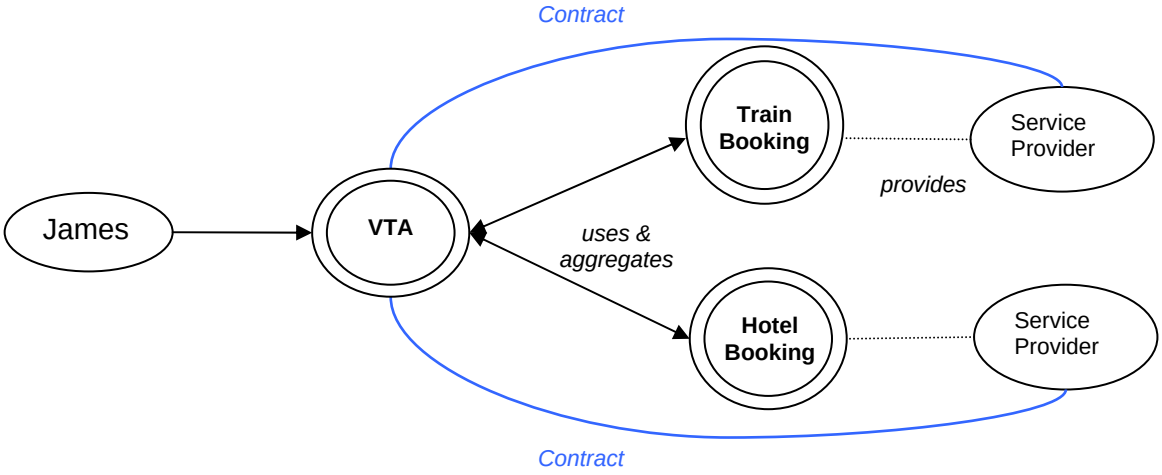

Figure 1: Virtual Travel Agency scenario

In the virtual travel agency example introduced above various end-user ticket purchasing services need to communicate with the booking service of the Austrian railway company in a strongly time- and reference-coupled manner. This implies in particular that [Krummenacher et al., 2005]:

(1) the booking service is expected to maintain connections to an arbitrarily high number of end-user services at the very same time, a situation which imposes high scalability constraints.
(2) the services involved in the scenario need to know each other and share a common representation of the exchanged data. Due to the fact that the content of the information is hidden in the body of the SOAP messages and is not addressed as an explicit URI-identified Web resource, the interacting Web Services can not take advantage of the Web-specific security mechanisms as long as they do not understand the XML schemas used to represent the data. Achieving an agreement on the representation format and its meaning is assumed to take place prior to the inter-process communication, implying in this case that the end-user services are presumably expected to use and understand the semantics of the data formats applied by the railway agency service.

The Linda coordination language [Gelernter, 1985] foresees a communication mechanism based on a logically shared memory called "tuple space". We expect that semantically enabled tuplespaces can offer an infrastructure that scales conceptually on an Internet level. Just as Web servers publish Web pages for humans to read, tuplespace servers would provide tuplespaces for the publication of machine-interpretable data. Providers and consumers could publish and consume tuples over a globally accessible infrastructure, i.e., the Internet. Various tuplespace servers could be located at different machines all over the globe and hence every partner in a communication process can target its preferred space, as is the case with Web and FTP servers. This highlights many advantages for providers and consumers. The providers of data can publish it at any point in time (time autonomy), independent of its internal storage (location autonomy), independently of the knowledge about potential readers (reference autonomy), and independent of its internal data schema (schema autonomy) [Krummenacher et al., 2005]:

- **Space autonomy**: Producers and consumers can run in completely different computational environments as long as both can make access to the same event service, i.e., space-wise the processes are completely de-coupled
- **Reference autonomy**: the processes that interact through an event service do not need to know each other (anonymous). The notifications published by publishers are accessed by consumers indirectly. In general, notifications do not include references to concrete consumers, and similarly consumers do usually not include specific references to producers.
- **Time autonomy**: the processes that interact through an event service do not need to be up at the same time (asynchronous). In particular, producers might generate some notifications while related consumers are not connected with the event service, and the other way around, consumers might get notifications while the original producers are not online.
- **Semantic autonomy**: semantic persistent spaces provide a consensual conceptualization and representation of the data published in each space. This approach facilitates the integration of data and processes.

In terms of the virtual travel agency use case previously introduced, a tuplespace infrastructure would imply the following scenario: the travel agency services would publish the travel information independently of any time and knowledge about the potential purchasing services and their internal data storage. In the same manner, the end-user services would subscribe to the information the travelers are interested in. The end-user services would be notified if new traveling data matching their requests is available. Although the inclusion of persistency, anonymity and asynchrony in the communication between Semantic Web Services are clear advantages, the VTA example raises interesting issues for renewed research efforts in tuplespace computing, e.g. since customers (James), traders (VTA) and service providers (hotel and train companies) publish information into the same tuplespace, how do we limit accessing James' tuples to only the VTA service?

In this document we introduce recent approaches in the field of semantic tuplespace-computing, which are expected to provide a feasible alternative to current Web Services technologies and the aforementioned problems. We give an overview of four semantic tuplespace platforms in Chapter 2, describing the most elaborated ones, namely CSpaces and Semantic Web Spaces, in more detail. The results of this survey are compiled in Chapter 3 into a unified conceptual framework for tuplespace computing on the Semantic Web, which subsumes the most important functional dimensions commonly identified in the analyzed proposals, as well as preliminary architectural decisions towards their implementation. The application of the tuplespace framework in the area of Web Services is elaborated in Chapter 4, while conclusions and future work are summarized in Chapter 5.

2 Overview of current proposals

2.1 sTuples

sTuples [Khushraj et al., 2004] has been developed as part of the Pervasive Computing work at the Nokia Research Center. Given the particular characteristics of pervasive environments, i.e. the heterogeneity and dynamics of multiple clients in the environment, the Semantic Web was seen as a solution to semantic interoperability issues, while tuplespaces were seen as a satisfactory middleware able to provide data persistence, as well as temporal and spatial de-coupling and synchronization. sTuples was built as an extension of Sun's JavaSpaces, which provides a centralized server and already extends the classical tuplespace model with field and tuple typing (based on Java's object-oriented model), Java objects as tuple contents, object-based polymorphic matching, transactional security and a publish-subscribe mechanism. It is also integrated with the Vigil framework for realising "Smart Home" scenarios in which mobile clients access home devices such as lights and consumer electronics over low-bandwidth wireless networks. Vigil provides distributed trust, access control and authentication services in the pervasive computing environment.

sTuples consists of three key extensions to the JavaSpaces platform:

- **Semantic tuples** extend the JavaSpace object-based tuple
- **Tuple template matching** is enhanced by using a semantic match on top of object-based matching
- **Specialized agents** reside on the space and perform user-centric services such as tuple recommendation, task execution and notification.

A **semantic tuple** is a JavaSpace object tuple which contains a field of type DAML+OIL Individual. This field contains either a set of statements about an instance of a service, or some data or an URL from which such a set of statements can be retrieved. Semantic tuples can be either data tuples or service tuples, depending on whether they contain semantic information provided by a service/agent or are advertising an available service (such as controlling a light or the volume of a television set). Both categories can be further refined in an ontology of semantic tuple types.

A **semantic tuple manager** is in charge of managing all interactions in the space concerning semantic tuples (i.e. insertion, reading and removal). When a semantic tuple is added to the space, the DAML+OIL statements it contains are extracted and asserted in the space's own knowledge base. The system checks that the statements are valid and that the knowledge base remains consistent. Likewise, when a semantic tuple is removed from the space, the statements that it contains are retracted from the knowledge base.

A **semantic tuple matcher** carries out the matching of templates to semantic tuples. Reasoning capabilities are provided by RACER, a Description Logics reasoner. A semantic tuple template, unlike the usual Linda approach of actual and wildcard values, is

a semantic tuple whose DAML+OIL individual-typed field draws upon a dedicated 'TupleTemplate' ontology. A set of statements using this ontology can be interpreted by the matcher as a semantic query upon the statements in the space's local knowledge base. However, due to the increased complexity of different DL based queries, the matcher performs its matching through a series of steps of increasing complexity.

1. the statements are validated against the TupleTemplate ontology so that invalid queries are immediately rejected
2. the candidate semantic tuples are selected by matching their tuple type (e.g. LightService as a subclass of ServiceTuple) against the value of the *hasTupleCategory* property in the query
3. RACER reasons over the set of candidate tuples so that inferable facts can be available (e.g. all classes that an individual belongs to through subsumption)
4. the tuple template contains different *TupleFields* which express desired or undesired field types and values. An exact match occurs when a semantic tuple is found which contains all desired tuple fields (in terms of the expressed type and possibly value) and does not contain any undesired tuple fields.
5. If there were no matches, and subsumption matching was requested in the tuple template, then the subsumption of field types is also taken into consideration in searching for a match.
6. If there were no matches, and plugged-in matching was requested in the tuple template, then plugged-in tuples will be matched.
7. Otherwise there were no matches and no tuple is returned.

Matching tuples will be weighted based on the degree to which they match the template (e.g. if all desired fields are matched, the extent to which undesired fields are not present). The matched tuple with the highest weight is selected to be returned to the client.

Finally, **specialized agents** reside in the space and offer added functionality to the user by abstracting typical user functionality needs and hence simplifying client interactions. In general, clients continue to interact with the Service Managers in the Vigil framework which mediate between the clients and the available services in the network. New services now register themselves in the system by passing a Service Tuple instance to the manager containing a service id, the DAML+OIL instance describing the service, a free text description, a service icon, a limit of the number of threads the service can support, a lease (specifying the duration the service remains active) and a location dependency indicator. Likewise, data from clients or services are passed as Data Tuple instances to the manager and contain a unique id for the tuple producer, a DAML+OIL instance containing the data to be shared and a list of subscribers to that object. In Vigil, the Service Managers are arranged in a tree-like hierarchy and each has its own space and specialized agents.

The *tuple recommender agent* allows a client to register its interests with a Service Manager using a pre-defined preferences ontology. The agent can monitor the space for

any services or data that match the interests of the client. If no matches are found at the time of the request, a notification request is registered with the space for a specified time period for any matching tuples that may be added to the space.

The *task execution agent* acts as a proxy for the user. The client registers tasks with the manager using a dedicated task ontology. Matching service tuples are retrieved and subscribed to, and commands are sent to the service as specified in the task ontology instance (e.g. switching a light on or off). The service response can also be captured (if specified in the task ontology instance) to be returned to the client or passed to another service (in the case of composite tasks).

A *publish-subscribe agent* dynamically delivers data to users that have subscribed to it. A data tuple is written to the space that is meant to be shared by multiple clients. A client requests data tuples of a particular type by using the tuple template ontology. The agent will find a matching data tuple and add the requester to the tuple field containing the list of subscribed users.

In summary sTuples extends JavaSpaces to share DAML+OIL instances in tuple fields for the purposes of supporting the semantic interoperability of heterogeneous and dynamic clients in a pervasive computing environment. Matching is extended by using the RACER reasoner to semantically match on DAML+OIL statements. Queries are formed using a dedicated ontology which allows specifying the desired tuple type as well as desired and undesired tuple fields and their values. Finally, a set of agents exist in the space to perform specialized tasks like recommending tuples according to a client's interests, executing common tasks through atomic or composite service calls and enabling clients to subscribe to specific types of data being shared through the space.

sTuple's future work originally included adding automatic learning capabilities to the space (e.g. identifying common tasks that can be abstracted by the task execution agent) and migrating from DAML+OIL to OWL. However, at the time of writing of this deliverable there is no evidence that any further activity in these directions is taking place in sTuples. Hence sTuples remains an interesting and informative 'first attempt' at a Semantic Web-enabled tuplespace but our analysis will continue by focusing on more recent activities in this area upon which our work can also have an influence.

2.2 Triple Space Computing

Triple Space Computing (TSC) [Fensel, 2004] has been recently introduced as a possible solution to the current situation in the field of Web Services. Starting from the observation that Web Services do not follow the Web paradigm of '*persistently publish and read*', [Fensel, 2004] proposes to follow exactly this paradigm for the communication of data between software systems across the Internet by means of tuplespaces. Triple Space Computing extends tuplespace computing [Gelernter, 1985], a simple and flexible coordination mechanism, using RDF as the formalism for describing the content of tuples in a space. Instead of a flat and simple data model in which tuples with the same number of fields and field order but different semantics cannot be

distinguished, [Fensel, 2004] proposes the use of RDF ([Klyne and Carroll, 2004]) to overcome this problem and create a natural link from the space-based computing paradigm into the Semantic Web.

[Bussler, 2005] and [Martin-Recuerda and Sapkota, 2005] extend the work of [Fensel, 2004] in different directions. [Bussler, 2005] focuses on defining a minimal architecture for the Triple Space Computing. The essential elements of this architecture are briefly defined as follows [Bussler, 2005]:

- ✓ **Storage Object.** The objects that Web Services write and read are RDF triples as defined in [Hayes, 2004]. As a difference with [Hayes, 2004], triples are uniquely identified through URIs [Berners-Lee et al., 2005]. This means that each triple in any triple space is uniquely marked and can be distinguished from all the other triples by its unique URI. In this way triples become quads [MacGregor and Ko, 2004].
- ✓ **Triple Space Clients**. A triple space client writes and reads triples in parallel or sequentially. Clients are therefore not distinguishable from the viewpoint of a triple space. Every client can read and write triples according to their security rights.
- ✓ **Triple Space**. A triple space is a virtual concept implemented by triple space servers. A triple space has to be part of one implementation, but one implementation can host many triple spaces. The relationship is one-to-many between a triple space server and (virtual) triple spaces. Each triple space within a triple space server has to be distinguished so that triples are forwarded to the particular triple space that the writer indicated when invoking the write operation. A triple space is identified through a unique URI. Thus, triples are written and read with this URI as triple storage location.
- ✓ **Triple Space Transfer Protocol (TSTP)**. The triple space transfer protocol is used between TSC clients and triple space servers to initiate the operations of writing and reading triples. A simple implementation approach is to map the TSTP protocol to the HTTP protocol. In this case there is no native implementation of it; however, this approach has the benefit of using a proven and Internet-scalable technology.
- ✓ **Minimal Triple Space API** (table 1). Bussler proposes a minimal Triple Space API for Triple Space clients and servers. Clients can write and read single or multiple triples in a concrete Triple Space. Servers can execute basic administrative operations like create a new Triple Space, delete the content of a Triple Space and delete the Triple Space itself.
- ✓ **Triple Space Server**. A triple space server may host arbitrarily many triple spaces. TSC clients are not aware of triple space servers, but only of virtual triple spaces. A Triple Space Server has the following four components:

  → *Storage component*. The storage component stores the triples in form of relational databases, file systems, RDF databases, persistent queues, etc.

→ *HTTP communication component*. This component receives HTTP calls that implement the TSTP protocol. Each invocation -- a write or a read -- is forwarded to the TSTP operation component.

→ *TSTP operation component*. This component is responsible for writing and reading triples.

→ *TSC server operations*. The triple space server implements the write and read operations for triples as well as the error handling mechanisms. Furthermore, it implements the server operations for creating, deleting and emptying triple spaces.

Table 1. Minimal API for Triple Space Computing according to [Bussler, 2005]

API call and description		
API Client		
Void	**write**	**(Set triples|triple)**
Write one or more triples in a concrete Triple Space identified by a URI.		
Set triples|Triple|Error	**read**	**(Set URIs| URI)**
Return the first "quad" (set of "quads" or error) that has the same URI (or set of URIs) stored in a concrete Triple Space identified by a URI. The quads matched are not deleted from the Triple Space, and the read operation is not blocking.		
API Server		
Boolean	**create_triple_space**	**(URI)**
Create a triple space with URI as an id		
Boolean	**delete_triple_space**	**(URI)**
Delete the triple space identified by URI		
Boolean	**empty_triple_space**	**(URI)**
Delete the content of the triple space identified by URI		

[Bussler, 2005] agrees that the minimal architecture proposed is too simple even to be useful. Thus, he proposes in his technical report further extensions of the initial "minimal Triple Space Computing architecture" proposal. The following list presents briefly those extensions:

- ✓ **Rich semantics for read operations**. Instead to retrieve triples using their URI, Bussler proposes to extend the functionality of read operations to support a query language particularly tailored for RDF triples. Another extension for read operations, already contemplated by the classical Linda "out" operation, is to delete the triple after the read operation conclude successfully.
- ✓ **Rich semantics for write operations**. Write tuples specifically addressed for concrete readers. This feature is against the basic principle of tuplespace computing that decouples from references, but Bussler justifies this for eCommerce purposes or for simulating queues in distributed applications. Semantic Web Spaces addresses the same problem using contexts, and CSpaces proposes to split the virtual persistent space into shared and individual CSpaces with restricted access rights.
- ✓ **Destruction operations**. Bussler proposes that triples will be destroyed by the server when a concrete number of reading operations are performed on a triple, or when a time deadline is reached.

- ✓ **Constraints definition**. Bussler suggests that it would be interesting to have the possibility to define constraints that ensure consistency of the information stored in a Triple Space.
- ✓ **Transaction support.** According to Bussler, transactions are necessary if several set-oriented operations should be protected from lost updates. In this case, allowing clients to delineate the transaction boundaries using dedicated transaction operations are recommended. A triple space server should be able to restore one of its Triple Spaces to a state in which the set of written operations under an aborted transaction had never been executed. Bussler suggests to evaluate Web Service transaction protocols like the one discussed in [Cabrera et al., 2004a; Cabrera et al., 2004b; and Cabrera et al., 2004c].
- ✓ **Ontology definition.** In order to increase the capability of a Triple Space to express semantics, Bussler suggests to include the ability for triple space servers to store ontologies.
- ✓ **Access Security.** Bussler proposes that writers can specify access control to restrict the access to concrete triples to a selected group of potential readers. On the other hand, triple space servers should restrict writing capabilities to specific writers.
- ✓ **Transmission Security.** Instead of encrypting the information stored in a Triple Space, Bussler recommends the use of a secure communication channel like in HTTPS [Rescorla and Schiffman , 1999].
- ✓ **Non-Repudiation**. In eCommerce domains, a mechanism must be put in place that allows both, the sender and receiver, to proof that the message sent and received is the original one. Triple Space can provide the means to hold a copy of the message. In a case of dispute between sender and receiver, the third party's copy will be the determining factor.
- ✓ **History and Archive**. Bussler recommends that triple space implementation should provide a history functionality that ensures that every 'movement' of triples, be it writing them, reading them, deleting them or updating them is recorded in the history.
- ✓ **Location Directory.** Bussler suggests the inclusion of a search engine that stores all Triple Spaces and a short summary of the contents of each Triple Space. This infrastructure will help clients to identify potentially interesting triple spaces.
- ✓ **Versioning**. The capacity of store versions of a Triple Space is closely related with the transaction support for reading and writing operations. A version mechanism ensures that all prior data can be available later on.

On the other hand, [Martin-Recuerda and Sapkota, 2005] extends [Fensel, 2004] and [Bussler, 2005] with a richer coordination mechanism based on the combination of tuplespace computing and the publish-subscribe paradigm that decouples the processes involved in a communication in a new orthogonal dimension: ***flow decoupling***.

Participants are not blocked while producing/receiving notifications. Consumers can receive a notification while performing some concurrent activity (i.e. through a 'callback'). Producers can produce notifications continuous with their execution flow. In other words, the main flows of producers and consumers are not affected by the generation or reception of notifications.

Furthermore and following [Bussler, 2005], TSpaces [Wyckoff, 1998] and JavaSpaces[1] proposals, [Martin-Recuerda and Sapkota, 2005] includes also transaction support as a part of the coordination model. In particular, [Martin-Recuerda and Sapkota, 2005] stresses the use of a distributed transaction model in which transactions involve potentially distributed resources (i.e. spaces in TSC) and are usually based on a two-phase commit protocol. A transaction manager (TM) is responsible for coordinating a transaction by coordinating one or multiple resource managers (RM). A RM is a component which allows transactional access to some resource. Applications (AP) communicate with both the transaction manager and resource managers. Finally, [Martin-Recuerda and Sapkota, 2005] studies the suitability of Triple Space Computing as a coordination model for Semantic Web Services and object components. The report describes in detail how Triple Space Computing can benefit WSMX [Zaremba and Moran, 2005], an execution environment for the dynamic discovery, selection, mediation and invocation of Semantic Web Services described using WSMO (Web Services Modelling Ontology, [Roman et. al., 2005]). Table 2 gives an overview of the API proposed by [Martin-Recuerda and Sapkota, 2005] for Triple Space Computing and based on [Martin-Recuerda, 2005].

Table 2. API for Triple Space Computing according to [Martin-Recuerda and Sapkota, 2005]

API call and description		
Void	**write**	**(set triples, URI ts)**
Write one or more triples in a concrete Triple Space identified by a URI.		
Triple	**take**	**(Template t, URI ts)**
Return the first triple (or nothing) that match with the template (that can be expressed using a formal query language[2]) and delete the matched triple from a concrete Triple Space ts		
Triple	**waitToTake**	**(Template t, URI ts)**
Like take but the process is blocked until the a triple is retrieved		
Triple	**read**	**(Template t, URI ts)**
Like take but the triple is not removed		
Triple	**waitToRead**	**(Template t, URI ts)**
Like read but the process is blocked until the a triple is retrieved		
Set	**scan**	**(Template t, URI ts)**
Like read but returns all triples that match with template t		
Long	**countN**	**(Template t, URI ts)**
Return the number of triples that match template t		
URI	**subscribe**	**(URI agent, Template|Query t, Callback c, URI ts)**
A consumer (agent) expresses its interested on triples that match with template t in a concrete Triple Space. Any time that there is an update in the Triple Space, the subscriber receives a notification that there are tuples available that match the template. The notification is executed by calling a method/routine specified by the		

[1] http://java.sun.com/developer/products/jini/index.jsp
[2] Currently we are studing the possibility to use SPARQL and RDFQL.

API call and description
subscriber. The operation returns an URI that identifies the subscription.
Set unsubscribe (URI agent, Template|Query t, Callback c, URI ts)
A consumer (agent) deletes its subscription, and no more related notifications are received. The operation returns a set of URIs of subscriptions deleted
URI advertise (URI agent, Template|Query t, URI ts)
A producer shows its intention to provide tuples that match t. Advertisement provides information to the system that can be used in advance to improve the distribution criteria of data and participants. The operation returns an URI that identifies the advertisement created.
Set unadvertise (URI agent, Template|Query t, URI ts)
A producer shows its intention to do not provide more tuples that match t. The related advertisements are deleted, and the operation returns a set of URIs deleted.
URI getTransaction (URI ts)
Ask the TSC infrastructure to create a new transaction and returns its id as a URI.
Boolean beginTransaction (URI txn, URI ts)
Identify the beginning of a set of instructions executed under a concrete transaction (identified by a URI). Several processes can execute instructions under the same transaction, and only those processes can see the changes produced in the space before the transaction is committed.
Boolean commitTransaction (URI txn, URI ts)
Make permanent a set of changes defined inside of a transaction txn.
Boolean rollbackTransaction (URI txn, URI ts)
Undo a set of changes defined inside of a transaction txn.

The current status of the work presented by [Fensel, 2004; Bussler, 2005; and Martin-Recuerda and Sapkota, 2005] is still in a very early state, and important elements like the organizational model and the security and trust model are not well addressed.

2.3 Semantic Web Spaces

Semantic Web Spaces [Tolksdorf et al., 2004; Tolksdorf et al., 2005a; and Tolksdorf et al., 2005b] has been proposed by the Freie Universität Berlin. It was originally envisaged as an extension of their XMLSpaces work, an implementation of a tuplespace platform which extended the Linda co-ordination model so that tuple fields could also contain XML documents and match templates based on XPath expressions or other XML Query forms. In the proposed next stage, a RDFSpaces platform would extend the Linda co-ordination model to support the exchange of RDF triples as tuples, with matching based on RDFS reasoning capabilities. As this platform was seen as the first step in modelling tuplespace-based communication for the Semantic Web stack (and hence there would be OWLSpaces, RuleSpaces, ProofSpaces and so on) the work has been named Semantic Web Spaces.

A conceptual model has been drawn up [Paslaru-Bontas et al., 2005a; and Tolksdorf et al., 2006] in which the necessary extensions to the traditional Linda co-ordination model were considered to support a tuplespace exchanging Semantic Web information. These extensions can be split into four categories:

- ✓ new types of tuples
- ✓ new co-ordination primitives
- ✓ new matchings

- ✓ new tuplespace structure

A further report [Paslaru-Bontas et al., 2005b] considers the use of Semantic Web Spaces as a middleware for distributed, concurrent Semantic Web applications, choosing an ontology repository scenario to illustrate its operation.

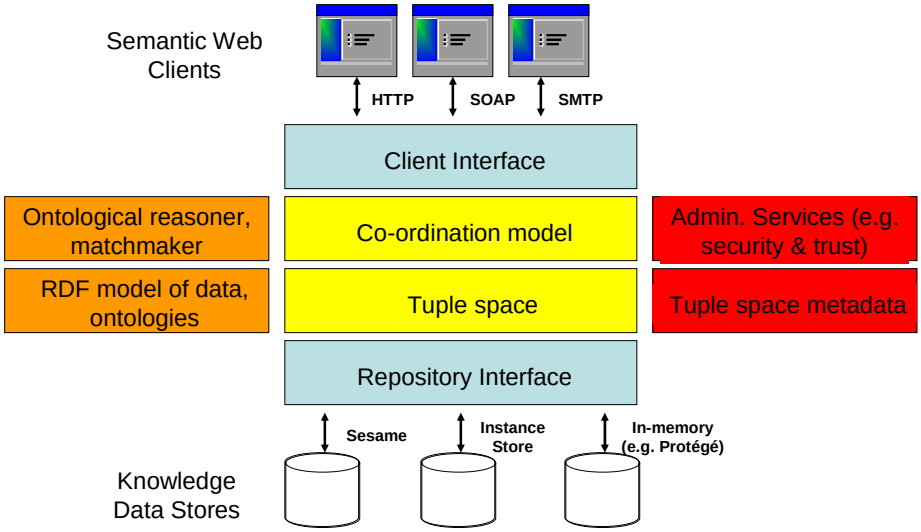

Figure 2: High level architecture of Semantic Web Spaces

Figure 2 shows a high level architecture of the Semantic Web Spaces. Like all Linda-based systems, the central components are the Linda co-ordination model and the tuplespace as a shared data space for tuples. In the Semantic Web Spaces we extend the core architecture with a reasoning component for interpreting ontologies according to their formal semantics (and drawing inferences, checking satisfiability etc) as this is out of the scope of the Linda paradigm. Accordingly, the tuplespace is extended to support building a semantic view upon the tuples (i.e. construction of a RDF graph model from RDF data stored in the tuplespace) and association of RDF statements with the ontologies they reference.

Additionally, it extends the component handling the co-ordination of processes with modules to fulfill different administrative services as are determined as requisite in a Semantic Web middleware. We consider here e.g. issues of security and trust.

This is complemented with a set of metadata for the tuplespace itself, according to an ontology we define for describing a tuplespace and the tuples that it contains. This ontology provides concepts for expressing security and trust policies, hence allowing for an ontology-based approach to organizing, initializing and configuring these extension

modules. Further on, the ontology explicitly describes the structure of the space (e.g. whether sub-spaces are allowed) and the supported matching templates. Finally, as the system is foreseen as a middleware platform, it should be independent of the underlying implementations of the different computer systems that the system must interact with. This necessitates interfaces to isolate the system kernel from the heterogeneity of both the clients which communicate with the system and the backend storage solutions which realize the physical storage of the information represented in the logical memory of the tuplespace.

We briefly outline the extensions that are proposed by the conceptual model in order to make the co-ordination model and tuple space Semantic Web compliant.

- ✓ **New types of tuples** – a RDFTuple is defined which contains four fields which take URIs as values: subject, predicate, object and ID (the object field can also take a literal value, i.e. a XML Schema datatype). Each field is also typed by an URI (the ID field is a RDF ID). These URIs represent instances and their classes, respectively, in a RDF model. Hence every RDFTuple represents a single RDF statement together with a unique ID for that tuple. Special consideration is taken for representing blank nodes, containers/collections and reification.

- ✓ **New co-ordination primitives** – a criticism of Linda has been that the semantics of the co-ordination primitives (`in, out, rd`) were never formally defined by the creators of Linda. When working with Semantic Web data it is important that the set of co-ordination primitives are clearly defined. In Semantic Web Spaces, two levels of interaction are defined: the data level, where tuples contain data without any formal meaning, and the information level, where RDFTuples are recognized as being special data structures that express formally defined knowledge about concepts. RDFTuples are handled also at two levels: in terms of the abstract syntax and in terms of the formal semantics. These three levels of co-ordination provide an increasing level of expressivity at an increasing cost in computability. Table 3 lists the co-ordination primitives of Semantic Web Spaces.

Table3: Co-ordination primitives of Semantic Web Spaces

API call and description		
Data Level		
out(tuple)	returns Boolean	Classical Linda out
rd(template)	returns tuple	Classical Linda rd
in(template)	returns tuple	Classical Linda in
Information Level (RDF Syntax)		
outr(s,p,o)	returns Boolean	Only true if tuple is RDFTuple
rdr(s,p,o,id)	return RDFTuple	Only matches on RDFTuples
inr(s,p,o,id)	return RDFTuple	Only matches on RDFTuples
Information Level (RDF Semantics)		
claim(s,p,o,id)	returns Boolean	An out which only returns true if the RDFTuple(s) conform to all available (RDFS/OWL) ontologies
claim(subspace)	returns Boolean	
endorse(s,p,o,id)	return Subspace	A rd with semantic matching using available

		RDF/OWL reasoning
extract(s,p,o,id)	returns Context	Multiple read version of endorse - finds all matching RDFTuples and places them into a context
retract(s,p,o,id)	return Subspace	An in which does not remove a matched RDFTuple (which would be akin to negation) but replaces its <s,p,o> values with null values

Two terms from the table can be explained: subspaces are first class objects which encapsulate one or more RDFTuples and are used to express multiple statements in one operation (in terms of claim) or return RDF sub-graphs which may contain blank nodes (similar to Concise Bounded Descriptions); contexts are introduced in the description of the tuplespace structure.

- ✓ **New matchings** – while the data level considers all tuple content at a purely syntactic level (and hence can perform the usual datatype matchings such as string or URI equivalence) the information level introduces Semantic Web specific matchings using RDF/OWL-specific reasoners. In combination with available ontologies, RDFTuples introduced to the space can be checked for ontological conformance and template matches can be made not only against the actual RDFTuples in the space but also those which can be inferred, e.g. `subClassOf` and `subPropertyOf` statements allow matches to take place on the basis of subsumptive reasoning, i.e. any variable typed with `Class` A in a template can be matched to a constant typed in `Class` B in the respective field of a tuple if A subsumes B.

- ✓ **New tuplespace structure** – while the original Linda considered a single tuplespace, extensions have introduced multiple, nested and hierarchical spaces. The distributed and replicated Semantic Web Spaces are virtually partitioned using *contexts*, drawing on the concept of scopes [Merrick and Wood, 2000].

Clients may be allocated certain contexts, controlling their view upon the space to those tuples existing within their context. Contexts provide a simple form of access control, allowing clients to have private spaces as well as shared spaces with specific other clients. From the system perspective, they can be used to perform clustering (of RDFTuples which are related in some way) and hence to improve matching efficiency.

In addition, Semantic Web Spaces defines an ontology for describing the space itself. Thus it creates a meta-space of RDFTuples which explicitly represents the actual structure of the active Semantic Web Spaces. An instance of the Semantic Web Spaces ontology forms a queryable (and possibly editable) description of the space, including its permitted structure, supported tuple types and matching templates, and effective access and trust policies. The meta-model of the Semantic Web Spaces will contain all instances of tuples currently stored in the space (and hence provides for each the unique URI by which they can be referenced) and can store meta-

information relating to each tuple such as its author, insertion time, number of reads or current context. A part of the tuplespace ontology is shown in Figure 3.

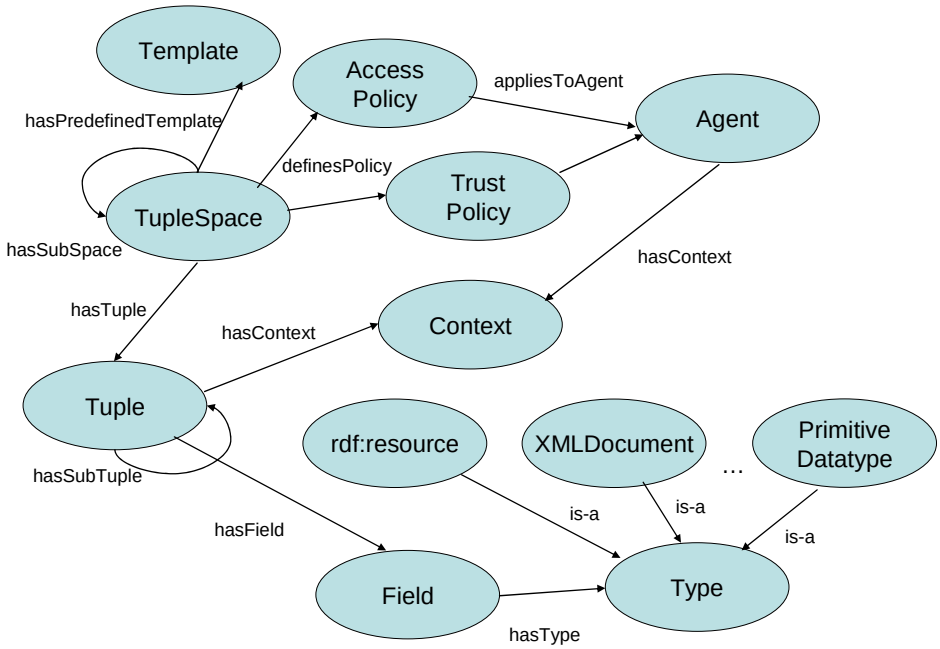

Figure 3: Ontology for Semantic Web Spaces

As the extensions to the core Linda model that are proposed to enable the co-ordination of Semantic Web knowledge reach to the fundamentals of the classical tuplespace paradigm, it was found that it would not be possible to build a Semantic Web Spaces as an extension of XMLSpaces and hence an implementation from scratch will be done.

2.3.1 Semantic data and organizational model

The semantic model of Semantic Web Spaces is to represent RDF information in dedicated tuples typed as *RDFTuple* and to consider that an agent has two views upon a tuplespace consisting of RDFTuples:

- ✓ A data view, i.e. viewing the RDFTuples as data-containing tuples according to the classical Linda model.
- ✓ An information view, i.e. viewing the RDFTuples as knowledge-containing tuples which form a RDF graph consisting of all of the statements expressed within the tuples.

This dichotomous view upon the tuplespace has guided the design decisions in Semantic Web Spaces, both conceptually and in terms of an implementation.

The organization model of Semantic Web Spaces is *contexts*. Contexts are an application of the idea of 'scopes' introduced in [Merrick and Wood, 2000]. Their usefulness is argued in improving scalability of open distributed Linda systems and enriching interaction patterns without expanding the number of co-ordination primitives. Rather than using multiple or nested tuplespaces, scopes *logically* partition the single tuplespace into arbitrarily overlapping *physical* subspaces. A scope can be considered to be a particular view upon a tuplespace in which a certain subset of the tuples in the global tuplespace can be seen.

Scopes are implemented in that they have names, and are created by passing that name to the tuplespace using a `newscope` primitive. The co-ordination primitives are extended to specify the scopes in which they are operating. An inserted tuple is associated to the scope attached to the insertion primitive. Tuple matching only sees the tuples in the scopes attached to the matching primitive. Merrick and Wood demonstrate how scopes can support the multiple read operation and atomic transactions. It can also be understood that scopes can reduce the complexity of large systems by restricting operations to a specific subset of the space.

In Semantic Web Spaces, we reinterpret the notion of scope for a tuplespace that represents Semantic Web information, i.e. statements that carry a truth value. Contexts represent an agent's view upon the Semantic Web Space at a certain time point, i.e. the knowledge seen as valid to that agent at that time. Both agents and tuples are associated with a set of contexts which may change over time, either through agent actions or system actions. The association of contexts to both agents and tuples can be represented in the tuplespace ontology and hence a specific agent's or tuple's scope can be queried over that ontology.

Contexts use URIs for identification and can be considered instances of the Context class of the tuplespace ontology. In other words, we allow them to be considered Semantic Web instances that can have information attached to them and be shared in RDF documents. Agents are free to create contexts, though the general Web guidelines for URIs should be considered (i.e. place the URI in a namespace owned by the agent). The system can also create contexts within its allocated namespace for specific purposes such as in the case of multiple read operations.

Tuples inserted into a space exist in the contexts to which the agent, at the time of the operation, is associated. Likewise, retrieval operations match only against tuples in the current contexts of the agent. Agents can remove and add contexts associated to them by retracting and claiming statements using the tuplespace ontology.

We also allow a context individual to be the join of contexts. This can be modeled ontologically by instantiating an anonymous class which is the `owl:intersectionOf` anonymous classes which contain the individual contexts. The effect is to say that a tuple exists in a joined context if it exists in all of the intersecting contexts.

Contexts allow agents to operate in subspaces of the global space which contain the tuples relevant to them. Hence it can also be considered how agent activities, or perhaps abstracted to system agents, could gather related tuples into specific contexts so that agents could choose to act within that context to perform specific tasks. One other use of contexts would be a form of privacy and access control. An agent could use a context to place tuples private to it, or share a particular context with a group of other agents protecting the shared tuples from any other interactions. Contexts permitted or not permitted to an agent or considered public or private in a part of the tuplespace can be expressed in the access policy stored in the tuplespace model. Hence it would be defined if an agent joining the space (following authentication) would have access to the global 'public' space (i.e. excluding those parts of the space which have been specified as private) or a certain context. Hence agents could search the Semantic Web Space for certain tuples and choose to operate within their contexts, or if metadata relating to contexts were available, query on that metadata for find relevant contexts (effectively a discovery mechanism). By partitioning the space, we improve scalability of the system and enrich interaction patterns without having to add complexity to the co-ordination primitives.

2.3.2 Coordination model

Semantic Web Spaces is based on and compatibly extends the Linda language and its tuplespace-based co-ordination model.

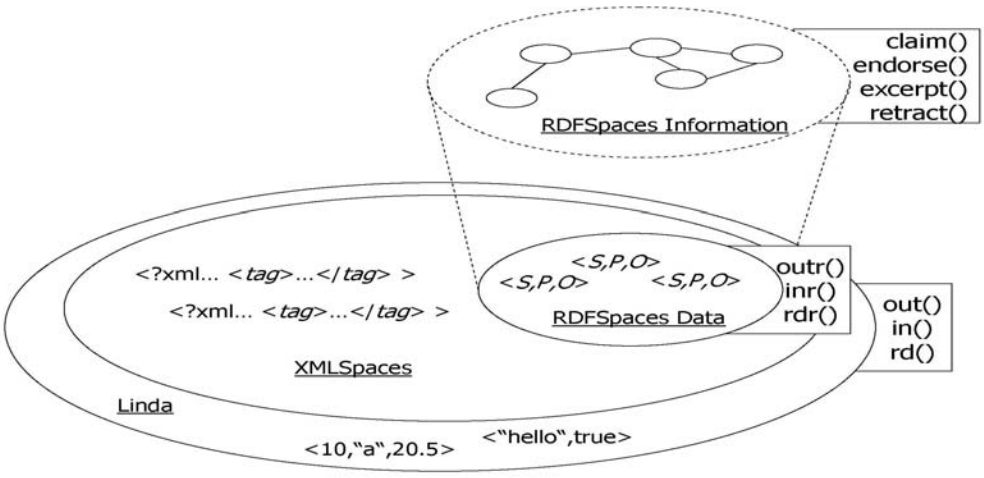

Figure 4: Coordination model in Semantic Web Spaces

The coordination language Linda has its origins in parallel computing and was developed as a means to inject the capability of concurrent programming into sequential programming languages. It consists of coordination operations (the coordination primitives) and a shared data space (the tuplespace) which contains data (the tuples). The tuplespace is a shared data space which acts as an associative memory for a group of agents. A tuple is an ordered list of typed fields. The coordination primitives permit agents to emit a tuple into the tuplespace (operation out) or associatively retrieve tuples from the tuplespace either removing those tuples from the space (operation in) or not (operation rd). Retrieval is governed by matching rules. Tuples are matched against a template, which is a tuple which contains both literals and typed variables. The basic matching rule requires that the template and the tuple are of the same length, that the field types are the same and that the value of literal fields are identical. Given the tuple (”N70241”,EUR, 22.14) - three fields containing a string, a pre-defined type (here, currency codes) and a float - it will match the template (”N70241”,?currency,?amount) and bind to the variables currency and amount the values EUR and 22.14 respectively.

The retrieval operations are blocking, i.e. they return results only when a matching tuple is found. In this way Linda combines synchronization and communication in an extremely simple model with a high level of abstraction.

Based on the conceptual distinction between the data and information view upon the RDFTuples of the tuplespace, the Linda co-ordination primitives are extended to distinguish agent interactions in the information view from those in the data view. This is itself built *upon the classical Linda model* and the XMLSpaces model (for the interaction with tuples containing XML data). However, at information level the original coordination mechanism is extended to contexts, as retracting and claiming operations apply only for the tuples associated to them. We illustrate this in the Figure 4 above.

2.3.3 Collaborative and consensus-making model

Semantic Web Spaces is modeled on the principles of the Semantic Web and hence does not specifically aim to impress any agreements onto agents interacting in the space as regards to content or semantics. In fact, it expects there to be a heterogeneity of content and semantics shared in the space (i.e. in the use of different ontologies and ontological models to represent knowledge about things) just as Semantic Web data will be exchanged between agents using different vocabularies, even to refer to the same things, and different models, even to express the same things. Rather, the idea of *mediation* is used to solve the problem of heterogeneity.

The usual Semantic Web communication is the point-to-point exchange of data between agents and knowledge sources, generally based on the Web communication model (HTTP GET/POST). While this (commonly called RESTful) style of communication does incorporate persistent publication of data (at an URL) this typically is not the case with Semantic Web information as agents interact on the basis of retrieving and updating individual statements or sub-graphs rather than entire files (which may indeed be

published at a known URL). Because of this, Semantic Web interaction is often based on querying over knowledge stored in an efficient storage platform such as a relational database (e.g. Sesame). Because of the point-to-point aspect of the communication the heterogeneity of content or semantics between the agent and the system it is communicating with must be resolved in advance, which requires both prior knowledge of the type of content/semantics required by the system and a means to map the agent's own content and semantic models to those required for communication with the other system. As a result, though the use of machine-understandable information is intended to support the automation of agent activities in the Web, dynamism of agent communication is only possible where system descriptions and means for mappings between content and semantic models are available.

Semantic Web Spaces is envisaged as being a type of *middleware* for the Semantic Web as it provides an interaction layer between agents and back-end storage, abstracting the access API to the *Linda co-ordination primitives* and permitting interaction at the individual triple level. Rather than point-to-point communication, the agents publish knowledge to and make queries over the space, de-coupling themselves in space, time and process from other agent systems which query their knowledge or provide knowledge to answer their queries.

Mediation becomes a task of the semantic matching algorithm applied within the space to retrieval operations. Semantic matching applies to the RDF graph formed by the RDFTuples of the (sub-)space, and is extended from syntactic matching in that it does not only consider the RDFTuples themselves (their fields, and their field types) but also the ontological knowledge stored within the space which defines classes and properties, and relations between them such as sub-classes and sub-properties (at the RDFS level) or transitivity or inverse properties (at the OWL level). As a result other statements are *inferable* from the RDFTuples which can potentially match a query that exist in the space.

Hence content mediation can take place through the provision of content mapping information and a semantic matching algorithm which seeks and applies this information when matching templates to RDFTuples. Mappings can be expressed in OWL (equivalence statements) but the Semantic Web Spaces shall be extendable to using rules (e.g. SWRL) or other semantic matching tools (e.g. those which use concept labels in combination with WordNet) to determine how a template according to some content model may relate to tuples using a different content model. Semantic mediation requires likewise the provision of semantic mapping information and a semantic matching algorithm which seeks and applies this information when matching templates. In this case the tuple/template using an alternative semantic model will need to be identified to the space (e.g. by extending the classes of tuple types to define a new type such as FLogicTuple…) and a component made available in the implementation that can handle the appropriate mapping (e.g. FLogic <-> RDF).

In conclusion, it is the aim of Semantic Web Spaces to support heterogeneity in agent communication by mediating between content and semantics within the space.

2.3.4 Security and trust model

Security is an important aspect of open distributed systems and trust is an important aspect in the sharing of Semantic Web information. Both shall be supported in Semantic Web Spaces. We consider both through an extension to its architecture (i.e. a dedicated component of the tuplespace platform monitoring interactions in the space) and through the co-ordination model itself.

An additional component is tasked with controlling security issues that do not relate to the co-ordination model itself, for example, authentication of agents and encryption of tuples. This includes the question of encrypting inside the tuples (the individual fields) or outside the tuples (the tuples themselves). We could use the reference architecture of [Bryce and Cremonini, 2001]. Here the component is called a 'reference monitor'.

Agents authenticate themselves by presenting a set of credentials to the reference monitor. If the credentials are accepted, an authentication token is presented to the agent. The communication between the agent and the space is associated to this token so that the space can authorize the agent interactions over the space. Rather than abstract security of the tuples to the communication protocol, agents could also receive a key from the system with which they encrypt their tuples. The security layer of the Semantic Web Space decrypts these tuples upon arrival, using the key associated with the agent which is identified through its authentication token.

One issue in security within the co-ordination model is who will have the rights to create and control access to contexts (and hence, tuples). In other words, there will need to be a top level access policy which controls who can create or change all other access policies (applied to agents or the space itself). This top level policy is controlled by the system administrator. Access policies should express at the very least:

- ✓ for agents, a list of contexts and spaces with their access rights for the agent (in, read, out operations)
- ✓ for a space, a list of agents with their access rights in each context (in, read, out operations)

Access might also be regulated not only by primitive but also by tuple content (e.g. accept only `out`'s of tuples matching a certain template).

Space access policies override agent access policies. In other words, access to some contexts may be restricted by the system administrator or the ability to restrict access to a context may be granted by the administrator to the context creator (this could be default). A context creator can then restrict access to a set of agents, regardless of what other agents say in their policies (note that this avoids the need for an '`invite`' type primitive,

and retains the Semantic Web approach of letting anyone say anything about anything, while ensuring that what is claimed is not always the case!). Unrestricted contexts may be added by agents themselves into their access policies.

Access policies could be modeled upon Access Control Lists (ACL) e.g. for a conference reviewing task there may be two contexts containing Papers and Reviews. The agent of the program chair would have the access policy `[(Papers,in),(Reviews,in)]` and the reviewer's agent has the access policy `[(Papers,read),(Reviews,out)]`, meaning simply that the program chair can destructively read papers and reviews from the space, while a reviewer can only non-destructively read papers and insert reviews into the space.

2.3.5 Architecture model

Originally Semantic Web Spaces was to be an extension of our XMLSpaces platform; however an initial prototype with RDF tuples and semantic matching demonstrated that the kernel of XMLSpaces was too tied to the XML data model; in order to add RDF support we would have to build upon the core of the platform from scratch. We foresee the use of a 'generic' Linda kernel, handling the classical Linda operations and datatype-based tuple matching, upon which the XMLSpaces and Semantic Web Spaces kernel would be built as extensions for tuples identified as being XML tuples (and hence handled as such in a XMLSpace) or RDF tuples (and hence handled as such in a Semantic Web Space). We would build upon a *classical* Linda system rather than a more complex implementation such as JavaSpaces or TSpaces in order to provide support for the classical Linda model within our platform and preserving the recognized benefits of the Linda approach: a small and simple set of co-ordination primitives to realize a powerful co-ordination model.

Semantic Web Spaces would be implemented as a dedicated extension to our classical Linda kernel. The kernel would be able to recognize RDF Tuples inserted into the space as well as RDFTuple-based retrieval operations (by the use of the dedicated primitives `inr/rdr/outr` or `claim/endorse/extract/retract`) and would pass these operations on to the Semantic Web Spaces kernel. Classical Linda operations (`in, out, rd`) would be handled by the core kernel, and hence the system would be *backwards compatible* to existing Linda interactions. The Semantic Web Spaces kernel would be extended on one side with a reasoning component and a semantic model of the knowledge in the space which is made available to the reasoner. On the other hand, a tuplespace model represents semantically the structure and characteristics of the Semantic Web Space and a Security and Trust component uses this model in policing agent access and activity on the space.

The implementation approach of Semantic Web Spaces can parallel much of the approach taken by XMLSpaces, such as the partial replication of contents, physical distribution of the space, and the addition of dedicated matching algorithms through the representation of the space as a XML document model (DOM) and the support for XML-

based querying. However, while XMLSpaces has the XML documents as field values, RDF Tuples represent RDF statements as the combination of their field values <subject, predicate, object>. Hence we support the representation of the space as a RDF graph and RDF-based querying.

RDF Tuples are physically stored as RDF statements in back-end storage solutions with appropriate scalability and performance. We do not check for inconsistency in the knowledge bases as this does not reflect the reality that inconsistency will occur on the Semantic Web. Rather, agents will have to expect and handle inconsistency in whatever way they decide.

RDF Tuple Templates are modeled as RDF queries (e.g. using SPARQL) which are executed across the RDF graph formed from all potentially matching RDF statements in storage (based on the context in which the interaction is taking place) using a reasoner component (different reasoners may be used to support different levels of matching e.g. RDFS, OWL-Lite, OWL-DL). Linda permits us to take the first matching tuple found. Given the need to use ontological information in semantic matching, ontological statements should be replicated across the space (or at least in that part of the space where statements using that ontology are found) so that they can be quickly retrieved and fed to the reasoner. Then as potential matches are found they can be immediately evaluated by the reasoner (e.g. to check for all valid classes through subsumption) and the first match returned.

The platform will also store a model of its own structure and characteristics as represented through a dedicated ontology. It is in this model that RDF Tuples receive URIs as IDs and both agents and tuples can be associated with contexts, for example. Hence the kernel will also check this model regularly during interactions to check interaction consistency, e.g. a tuple inserted into the space by an agent will be associated to the contexts in which the agent is active. Components in the space can be added which check contents of the tuplespace model and introduce additional mediation on agent interactions. A prototypical example which will be implemented in the Semantic Web Spaces will be that of Security and Trust (e.g. a component can access the server API and perform operations like (`in,permit`) or (`out,refuse`) according to the access rights found in the space). According to available access and trust policies expressed in the model agent interactions may be permitted, refused, preferred or given low priority. Likewise, as the system can organize the logical and physical storage of tuples in ways to improve performance and quality, another possibility would be to use trust policies to give more priority to more trusted tuples in the space (i.e. ensure a greater possibility of more trusted tuples being retrieved from the space than less trusted tuples). This could be later extended to notions of tuple self-organization [Tolksdorf and Menezes, 2003] and the 'fading away' of less trusted/used tuples in the space. At present, no concrete decisions have been made concerning the representation of the trust and access policies. Self-organization in tuplespaces will be tackled in the approaching TripCom[3] project.

[3] http://www.tripcom.org

Semantic Web Spaces will follow a component-based architecture. On top of the classical Linda kernel and the dedicated Semantic Web Spaces kernel (which adds support for the dedicated co-ordination primitives and the storage of RDFTuples, their representation as RDF graphs and the representation of the tuplespace model) we see the reasoner as a core component (so that we could 'plug in' different levels of reasoner as required) and other components as optional added functionalities that use the tuplespace model and access to the kernel API to allow extra mediation between agents and the tuplespace (e.g. for the implementation of security and trust).

2.4 Conceptual Spaces (CSpaces)

Conceptual Spaces (*CSpaces*) [Martin-Recuerda, 2005] was born as an independent initiative to extend Triple Space Computing [Fensel, 2004] with more sophisticated features and to study their applicability in different scenarios apart from Web Services (e.g. distributed knowledge management systems [Bonifacio et al., 2002a]). The original scope of CSpaces has evolved towards a new proposal for a conceptual and architectural model that can appropriately characterize most of the requirements and functionality that the Semantic Web demands. Although the Semantic Web research community have achieved significant results since 2001, several relevant questions are still open: how to keep coherence and consistency between the Web and the semantic annotations and how to annotate web pages that are not persistent (*dichotomy problem*); how to store and reason with the huge amount of semantic annotations expected to be published (*scalability problem*); how to organize and share semantic annotations and how to persuade current web users to create machine processable semantics (*publishing problem*); how to overcome conflicting terminology and conceptualizations defined by different ontologies (*heterogeneity problem*); how to ensure meaningful answers when the information stored is not consistent (*inconsistency problem*); how to guarantee that only a restricted amount of users can visualize and edit concrete semantic annotations (*security problem*); and how to guarantee validity and trustworthiness of the semantic annotations (*trust problem*).

With the so-called Web 2.0[4], the Web is becoming more dynamic and many of the web pages accessible are generated dynamically instead of being static. Thus, an approach to diminish the *dichotomy problem* is strongly required. Decreasing the amount of non-semantic data representation in the Semantic Web, and therefore, making machine processable semantics the prevalent representation formalism is the proposal that CSpaces promotes in order to minimize the dichotomy problem.

Just as the Web has been characterized by an abstract model called REST (Representational State Transfer) [Fielding, 2000] that is defined as a set of constraints (*client-server architecture, stateless, cache, uniform interface, layered system,* and *code-on demand*), CSpaces characterizes the Semantic Web around seven building blocks: *semantic data and schema model (knowledge container), organizational model,*

[4] http://www.oreillynet.com/lpt/a/6228

coordination model, semantic interoperability and consensus-making model, security and trust model, knowledge visualization model and *architecture model*:

- ✓ **Semantic data and schema model**. Defines a knowledge container, called a CSpace, in which data elements and their relations are described using a formal representation language[5] that includes a set of modeling primitives enriched with rules in order to build a logical theory. These knowledge containers also store relations, called annotations, between data objects and related external objects like documents and web pages. Also the relations with other knowledge containers are also included in order to facilitate interoperation among them. CSpaces can have associated access rights and maintain metadata information about themselves that include unique identifier, creator, list of members, etc.
- ✓ **Organizational model.** There are two types of CSpaces, Individual and Shared. Personal knowledge is stored by each agent in Individual CSpaces, and Shared CSpaces maintain knowledge that several users want to share using a common formal representation and a common conceptualization. The information stored in a Shared CSpace can appear in three different flavors: materialized view, virtual view [Ullman, 1997] and hybrid materialized-virtual view ([Alasoud et al., 2005] and [Hull and Zhou, 1996]). In addition, Shared and Individual CSpaces can be factorized and recombined in a collaborative manner in order to create new Shared CSpaces, and related CSpaces are connected by mapping and transformation rules that not only show explicitly common elements stored in different CSpaces, but also allow the execution of reasoning processes in a distributed fashion.
- ✓ **Coordination model**. CSpaces is a middleware infrastructure for applications and a cooperation infrastructure for humans. The coordination model is defined on top of mediated, semantic and persistent communication channels (Shared CSpaces) that represent at the same time knowledge containers. Thus the concepts of knowledge repository and communication channel become one, and messages can be described in a more compact manner, because message content can refer to ontological terms stored in the CSpace used for communication. The coordination model combines two metaphors: "*persistent publish and read*" (tuplespace computing) and "*publish and subscribe*".
- ✓ **Semantic interoperability and consensus-making model**. The role of Shared CSpaces is to promote that users of the Semantic Web reach consensus in the specification of a knowledge base and a set of mapping and transformation rules that explicitly indicate relations with other CSpaces. CSpaces aims to recover the original role of ontologies as *shared* and not only *formal specifications of conceptualizations*. The process to build these shared conceptualizations follows some principles of Human Centered Computer approaches[6], and the necessity of

[5] These formal specifications are not ontologies per se if they are not shared, following Gruber's definition of an ontology as a "shared conceptualization" [Gruber, 1993]

[6] *Human Centered Computer* can be defined as the development, evaluation, and dissemination of technology that is intended to amplify and extend the human capacities. "To be human-centered, a [computer] system should be based

users and applications to interact with each other will drive the creation of new Shared CSpaces.

- ✓ **Security and trust model**. The protection of private and restricted information stored and the inclusion of trusted mechanisms to guarantee the validity or trustworthiness of the information accessed are critical requirements for a successful development of a distributed information infrastructure.
- ✓ **Knowledge access model**. CSpaces promotes the minimization of the amount of syntactic data representation (current Web). Thus, an infrastructure that facilitates users to deal with machine processable semantics is required. An intensive use of knowledge access solutions based on the graphical representation of knowledge bases and mapping rules, controlled natural language[7], and natural language generation techniques are the mechanisms proposed.
- ✓ **Architecture model (*blue-storm*)**. CSpaces proposes a distributed and decentralized *hybrid* architecture based on P2P and client-server infrastructure in which a group of agents (human or not) store, read and share information. A client-server P2P configuration drives a two-tiered system. The upper-tier is composed by well-connected and powerful servers, and the lower-tier, in contrast, consists in clients with limited computational resources which are temporarily available. To facilitate the distinction between CSpace knowledge containers, the CSpace conceptual model and CSpace architecture, Martin-Recuerda called the architecture model "*blue-storm*"[8].

Given the absence of an appropriate reference that includes an up-to-date description of the current version of CSpaces, the author has considered appropriate to include in this document a more detailed description of this proposal.

2.4.1 Semantic data model

A **Conceptual Space** (CSpace) is a knowledge container defined as a set of tuples. In CSpaces each tuple has a well-defined structured that is represented by six fields:

<guid, fm, type, sguid, vguid, mguid>

Ideally, **fm** is a first order logical formula. However, restrictions imposed by applications and/or members of the CSpace can restrict **fm** to less expressive formalism (like description logics) or even can be just RDF triples or Topic Maps (enough for instances to describe annotations to resources). The field **type** identifies in which subspace belongs **fm**. Currently, there are six different subspaces defined: domain theory (dth), metadata (md), instance (ic), trust and security (ts), mapping and transformation rules (mtr), and

on an analysis of the human tasks that the system is aiding, monitored for performance in terms of human benefits, built to take account of human skills, and adaptable easily to changing human needs" (Flanagan, et al., 1997, p. 12).

[7] Subset of a natural language (for instance English) with a domain specific vocabulary and a restricted grammar in the form of a small set of construction and interpretation rules.

[8] Some logicians uses the term *blue* for information that is semantically described (blue information), and one of the aims of the CSpaces architecture is to facilitate the spread of machine processable semantics in the Internet (*storm*).

annotations (at). Each logical formula, **fm**, has associated another four identifiers[9]. The first one is a global unique id for the logical formula (**guid**, which can simplify reification, and make the code more compact). The second identifier is the global unique identifier of the CSpace where they were created (**sguid**, which attaches provenance to a logical formula), the third one is a version global unique identifier (**vguid**) that identifies each version of a logical formula[10], and the forth identifier (**mguid**) is the identifier of the member of the CSpace that stores the tuple. Given that each member of a CSpace has a reputation score, **mguid** can help to measure the degree of trustworthiness of each of the logical statements that are stored in a CSpace.

As it was mentioned before, a CSpace is subdivided in six different subspaces:

- ✓ **Domain theory** stores a logical theory which gives an explicit, partial account of a conceptualization [Guarino and Giaretta, 1995]. The set of logical formulas of this subspace exhibit some degree of semantic autonomy. Semantic autonomy represents a particular perspective of the world of an individual or group of agents (humans or not). This semantic autonomy is represented using a semantic specification that describes, organizes and classifies information according with an individual or shared interpretation [Bonifacio et al., 2002b]. The sub-space domain theory is associated with a *reasoning* sub-space that provides an *efficient* representation (in terms of reasoning performances) of the stored logical theory.

  Ideally, five modeling constructs can be used to build a domain theory: *concept, relation, function, axiom, and rule*:

  → A **concept** describes a set of objects or instances which share similar characteristics that are defined using attributes. Attributes constrain concept definition and are associated with a value type that can be any of the following atomic data types[11]: boolean, number (integer, float and natural), date, number ranges, text string, and set of text strings. Also an attribute value type can be another class, and can be specified as being reflexive, transitive, symmetric, or being the inverse of another attribute.[12]

  → **Relations** represent a type of association between concepts of the domain theory that is: $R \subset C_1 x C_2 x \ldots x C_n$ [Gomez-Perez, 2004]. Relations can have an

[9] Some ideas of the Pong data model [Rhea et. al, 2003] , the implementation prototype of OceanStore, are behind the design of the tuple model of CSpace.

[10] Since tuples can be replicated in other CSpaces, it is important to track provenance and version to verify that we are working with the latest version of the tuple. Thus, if the tuple is deleted in the source CSpace, we have to keep a tuple with an empty fm field.

[11] Based on XML Schema datatypes: http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

[12] When an attribute is specified as being transitive, this means that if three individuals *a*, *b* and *c* are related via a transitive attribute *att* in such a way: *a att b att c* then *c* is also a value for the attribute *att* at *a*: *a att c*. When an attribute is specified as being symmetric, this means that if an individual *a* has a symmetric attribute *att* with value *b*, then *b* also has attribute *att* with value *a*. When an attribute is specified as being the inverse of another attribute, this means that if an individual *a* has an attribute *att1* with value *b* and *att1* is the inverse of a certain attribute *att2*, then it is inferred that *b* has an attribute *att2* with value *a*. [DIP 1.7 2005]

arbitrary arity, and like concepts can also have attributes [Schreiber et al., 2002].

→ **Functions** are special case of relations in which the n-th element (return element) of the relation is unique for the n-1 preceding elements (arguments): $F(C_1xC_2x\ldots xC_{n-1})\text{-}> C_n$. The definition of the function includes in its body the expression that calculates the return value (C_n) in terms of the arguments (C_1, C_2, …, C_{n-1}) of the function [Gomez-Perez, 2004].

→ **Axioms** serve to model sentences that are always true. Frequently, they are used to model knowledge that cannot be formally defined by the other components. **Axioms** allow extending the domain theory with intentional information (i.e. the possibility to derive new information) [Gomez-Perez, 2004].

→ Like axioms, **rules** allow to extend the domain theory with intentional information. Rules also can be considered as sophisticated version of relations [Schreiber et al., 2002]. A rule has the form: *consequent* ← *antecedent*, where both are a conjunction of atoms, R(t1, …, tn) composed by variables and/or constants. The meaning of a rule can be informally described as: "*whenever (and however) the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold*" [Grosof, et al., 2003].

✓ **Metadata** provides information about the CSpace itself. Currently, the metadata is partially characterized by Dublin Core metadata specification, but in the close future, Ontology Metadata Vocabulary (OMV, [Hartmann et al., 2005]) will be also considered. A CSpace is characterized by a global unique identifier (**sguid**), creator (identified by a global unique identifier, **mguid**), version, type (individual or shared), content description, etc. The ***sguid*** or each CSpace and the *content description* (a set of ontological concepts and logical formulas that provide a brief description of the content of the CSpace) are stored in a Shared CSpace that plays the role of a global catalog. The *content description* is described by an upper level ontology of topics associated with the global catalog, and content description can be used as a filter that can reduce the scope of a query (read operation) in case that it is not possible to specify initially the target CSpace.

✓ **Instances** are used to represent elements or individuals of concepts and the values of their attributes in a domain theory. Instances can refer to documents and files stored in the Web (the link between CSpaces concepts and real world objects is made through annotations).

✓ **Annotations** define links between concepts and instances (topics) specified in each domain theory with information resources (occurrences). RDF[13] and Topic Maps[14] are two promising representation means to describe annotations. The **Resource Description Framework** (RDF) is a W3C recommendation to define a language for describing resources in terms of named properties and their values. All resources are

[13] http://www.w3.org/TR/rdf-mt/
[14] http://www.ontopia.net/topicmaps/materials/tao.html

identified using URIs and are described in terms of triples (subject, predicate and object).

On the other hand, **Topic Maps** are a new ISO standard for describing knowledge structures and associating them with information resources using *topics, associations*, and *occurrences* (TAO). [Park and Cheyer, 2005] has already suggest the potential of Topics Maps as a formalism that can connect an ontology layer with a resource (document) layer. Although RDF and Topics Maps look very similar, [Garshold, 2003] shows that there are important differences that technically make it difficult to merge Topic Maps and RDF into a single technology. The current state of the CSpace proposal leaves open the formalism used to describe annotations and only suggests to take into account both RDF and Topic Maps proposals.

- ✓ **Security and trust information** in open and distributed environments like CSpaces are intimately engaged. The *security and trust model* proposed for CSpaces combines features of three different and complementary models [Suryanarayana and Richard Taylor, 2004]: **credential and policy-based** trust management, **reputation-based** trust management, and **social-network-based** trust management. Thus, the subspace will store a list of members (authorized users and applications, each of them with a global unique identifier, **mguid**), roles that each member can play, access rights for each role, credentials, policies, binary trust relationships (opinion about other agent's trustworthiness), a local reputation of each member according to the opinions of the rest of the members of a CSpace. Furthermore, the global Shared CSpace that will play the role of global catalog will store also a global reputation score for each agent registered in the system.
- ✓ **Mapping and transformation rules** identify common ontological terms, relations and instances between related individual and shared CSpaces and facilitate interoperation among them. The mapping proposal will take into account past and current efforts in information integration and ontology interoperation. One of the first considerations that should be taken into account is the integration approach selected. There are two major approaches for integration of information [Alasoud et al., 2005]: (1) the data warehouse (DW) or materialized approach and (2) virtual approach (mediator based). In DW approach, huge amount of historic data is stored in the DW. In the virtual approach, on the other hand, the data is not materialized, but rather is globally manipulated using a virtual integrated view[15]. In this approach, the actual data resides in the sources, and queries against the integrated 'virtual' view will be decomposed into sub-queries and posed to the sources. DW is preferable when fast query response is required and when the data is not updated very often. On the other hand, restrictions of the data source owners to allow access to the information make a virtual approach the adequate solution in this scenario. A third approach which is a hybrid between fully materialized and fully virtual approaches inherits the advantages of both. [Alasoud et al., 2005]. Because a CSpace is also a persistent communication

[15] With some extensions, Google follows a DW approach. Periodically, the wrappers of Google retrieve all the information of the Web which is stored and indexed on Google servers.

channel, in many occasions CSpaces will be constructed as a materialized or hybrid view.

For modeling the virtual integrated view different approaches has been studied as a model for the integrated schema [Ullman, 1997], such as, Global as View (GAV), Local as View (LAV), and several combinations of both: Global-Local as View (GLAV), Both as View (BAV, [McBrien and Poulovassilis, 2003]) and BGLAV ([Xu and Embley, 2004]). In the GAV approach, for each relation R in the mediated schema, we write a query over the source relations specifying how to obtain R's tuples from the sources. The query processing in GAV is easy, because we need only unfold the definitions of the mediated schema relations, but this approach does not help much when the sources change or grow very often. In contrast, the LAV approach defines the mapping in the other way around; each relation in the data sources is defined in terms of a query over the integrated schema. This makes query processing more difficult, since now the system does not know explicitly how to reformulate the concepts in the integrated view expressed in the user query in terms of the data sources. On the other hand, changes or incremental growth in the sources will not lead to reconstruction of the integrated schema, and need only to modify the mappings. The GLAV (BAV or BGLAV) approach is the combination of the GAV and LAV approaches, and it consists in associating views over the global schema to views over the sources to get advantage of the benefits of GAV and LAV.
Apart of GAV, LAV and GLAV (BAV or BGLAV) proposals, other formal interoperation approaches for ontologies have been proposed [Serafini et al., 2005]: Distributed First Order Logic (DFOL, [Serafini et al., 2005]), C-OWL [Borgida and Serafini, 2003], Ontology Integration Framework (OIS, [Calvanese et al., 2002b]), DL for Information Integration (DLII, [Calvanese et al., 2002a]), and ε –connections [Grau et al., 2004].

A concrete proposal for mapping and transformation rules for CSpaces is not yet defined, and it is postponed as a future work.

2.4.2 Organizational model

Nowadays, there is a debate in the Semantic Web community about how ontologies, rules and alignment specifications should coexist. Mapping and merging ontologies have been the common proposals to deal with heterogeneity in the Semantic Web. [de Bruijn and Polleres, 2004] argues that it is difficult to create ontologies from a consensus process [Visser and Cui, 1998; and Uschold, 2000], and applications that rely on specific ontologies can become inoperative after a merging process. On the other hand, scalability is the main limitation of the mapping approach, because it requires $O(n^2)$ ontology mappings, where n is the number of ontologies [de Bruijn et al., 2004]. *Ontology islands* [de Bruijn and Polleres, 2004; and de Bruijn et al, 2004] combine several advantages of mapping ontologies (local ontologies are not deleted and keep compatibility with related applications) and merging ontologies (more scalable since we do not have to maintain one to one mappings between each ontology). On the other hand, [Cook and Brown, 1999; and Kotis and Vouros, 2003] suggest that a consensual process for the generation

of merged ontologies not only is a means to exchange knowledge, but also a means to generate new knowledge. Those experiences have influenced the organizational model proposed by CSpaces.

Two types of CSpaces have been proposed: Individual and Shared CSpaces. The former is a knowledge container defined by an individual that reflects his/her own perception of a concrete domain. The machine processable semantics stored in an Individual CSpaces can be private (only the owner of the space can access it), restricted (a limited number of individual can access it) or public (the information can be accessed without restriction). The combination of restricted and public data can be used to create Shared Conceptual Spaces. Shared CSpaces are conceptual spaces shared by several participants that have reached an agreement on how to represent semantically common terms and logical statements. This requirement is fundamental to facilitate interoperability between user and applications in the Semantic Web.

The CSpaces that act as sources of a new Shared CSpace are not deleted to avoid the necessity to update related systems/applications. Mapping and transformation rules between sources (individual and shared CSpaces) and new Shared CSpaces will be created and maintained to identify equivalent terms and to avoid the necessity to copy all instances to Shared CSpaces.

CSpaces can contribute to organizing and sharing knowledge in the Semantic Web using a bottom-up (from personal knowledge specifications to shared knowledge specifications) approach and also can encourage the use of ontologies as "*shared*" and not only *formal specifications of conceptualizations* [Borst, 1997]. Instead of centralized systems that force users to agree to a set of rules, schemas and data, CSpaces offers a distributed infrastructure where users can publish their own knowledge based on their own conceptualization. Common point of views, interests and interoperability requirements of users will drive to the creation of Shared CSpaces.

This approach is inspired in an earlier proposal called Distributed Knowledge Management [Bonifacio et al., 2002a; and Bonifacio et al., 2003] where its authors confirmed during the realization of several tests in real scenarios that users were more favorable to this kind of approach because it takes into account the different perspectives and understandings that users have about the world and more concretely about the information, processes and interactions of their organizations or working groups. The combination of Individual CSpaces can generate a new space shared by all these users. Shared CSpaces is built on top of a semantic data representation agreement of a group of users. Moreover, Shared CSpaces can be combined to generate bigger Shared CSpaces, or in other words, bigger knowledge repositories.

CSpaces can be viewed as leaves and shared spaces can be graphically considered as the branches and the trunk of a fictitious tree following a very similar organization proposed in CO4 (Collaborative construction of consensual knowledge bases) [Euzenat, 1995]. CO4 is an infrastructure enabling the collaborative construction of a Knowledge Base through the web. One of the main contributions of this approach is a proposal for organizing Knowledge Bases in a DAG (*Directed Acyclic Graph*) configuration. The leaves of the graph are called user Knowledge Bases, and the intermediate nodes, group

Knowledge Bases. Each group Knowledge Base represents the knowledge consensual among its sons (called subscriber Knowledge Bases) [Euzenat, 1995]. This organizational model is very appropriate for distributed and related knowledge containers, because cyclic references are avoided (critical for distributed queries). However, it is difficult to restrict the creation of new Shared CSpaces which do not follow this DAG organization model. The presence of a global catalog in which the metadata of each CSpace is published together with its dependencies with other CSpaces can help to identify cyclic dependencies, but the issue is currently open.

2.4.3 Coordination model: "publish, read and subscribe"

Two main goals have to be achieved by the Coordination Model of CSpaces: provide a simple and powerful coordination mechanism for applications; and offer a flexible and effective communication channel for human users that can compete with well-known communication infrastructures like email systems.

Thanks to the Web, humans can *persistently publish and read* information at any time stored on servers spread around the World. The "*persistent publish and read*" metaphor have been also applied successfully as a simple coordination model for parallel computing called tuplespace computing [Gelernter, 1985], and more recently to Semantic Web Services [Fensel, 2004].

Tuplespace computing [Gelernter, 1985] is a coordination mechanism in which synchronization and communication between participants take place through the insertion and removal of tuples to/from a common shared space. Shared, persistent, associative, transactional secure and synchronous/asynchronous communication is the main property of tuplespaces. However, tuplespace computing has two relevant drawbacks: it does not provide flow decoupling from the client side; and the tuples published in the space do not rely on any schema or well defined semantic representation [Fensel, 2004]. The interaction model provides time and space decoupling but not flow decoupling [Eugster et al., 2001]. A user who is interested in an update version of a concrete web page has to check periodically until the update is available (*flow coupling* from the client side). To improve this situation, applications/users (subscribers) can store subscriptions with a description of the data that they would like to get, and when the data is available, subscribers will receive a notification with the information requested.

On the Web, the "*client flow coupling*" is a consequence of the REST (Representational State Transfer) [Fielding, 2000] architecture style, and in particular, because resources are stateless. To overcome this limitation, the "*persistent publish and read*" metaphor has been extended by Martin-Recuerda ([Martin-Recuerda, 2005]) into "*persistent publish, read and subscribe*". The popularization of "weblogs[16]" (blogs or bloggings) in conjunction with the development of RSS (Rich Site Summary or Really Simple

[16] A website which stores miscellaneous notes updated regularly and published in chronological order.

Syndication, http://www.rss-specifications.com/) brings a new form of interaction for web-users based on content subscription[17].

Tuplespaces have the same limitation from the reader-side. In classical Linda, an application which wants to read a concrete tuple has to run a process that blocks until the data is available[18]. JavaSpaces[19] and TSpaces[20], concrete Java implementations of the tuplespace approach provide a simple *notification* mechanism to mitigate the problem. [Martin-Recuerda, 2005] claims that publish-subscribe technology [Eugster et al., 2001] can complement the classical tuplespace approach with a notification and subscription mechanism that allows a proper asynchronous interaction from the consumers/reader side. Together with the inclusion of a distributed transaction model, the CSpaces coordination model provides the basic means that software applications and components require for communication [Martin-Recuerda, 2005].

On the other hand, in tuplespaces it is not possible to know beforehand which is the format of the data that producers will use to publish information in the space, and therefore, there is no way to know which data format the consumers expect. An implicit agreement is expected, but in the Semantic Web where millions of users and applications will interact, these implicit agreements are not feasible. In addition, the lack of semantics in data represented by Web means and in most tuplespace and publish-subscribe implementations limits the ability of search engines to provide accurate answers. Since CSpaces will provide rich schema specifications for the data stored, the coordination model based on the integration of the two technologies (tuplespace and publish-subscribe) have to be extended to support machine processable semantics[21]. For instance, in Triple Space Computing [Fensel, 2004] the data published and accessed is represented by RDF triples. [Li and Jiang, 2004] proposes an equivalent approach for event-based systems using DAML+OIL to express more accurately subscriptions and to improve event filtering mechanisms. However, the use of languages likes RDFS, OWL (DAML+OIL) and FOL (First Order Logic) is only part of the solution. The same terms can be described using OWL, but they can belong to different conceptualizations, and thus, they can have different meanings. To overcome this situation, two possible approaches can be taken: the use of mediators between applications and the communication channel, and the use of mediated persistent communication channels (the purpose of Shared CSpaces).

CSpaces integrates tuplespace and publish-subscribe operations, transaction support and semantic data specification in a new coordination model. The coordination model API for CSpaces is very similar to Triple Space (according to the proposal of [Martin-Recuerda and Sapkota, 2005] described on Table 2). However, CSpaces does not deal directly with triples but with tuples (please refer to section 2.4.1), and the API also has to take into

[17] RSS is still based on polling, but it is invisible to the end-user. Users subscribe, but their RSS client is just polling the webserver every x minutes.

[18] There have been extensions with non-blocking read operations. In this case the application must periodically read from the space until the data is found.

[19] http://java.sun.com/developer/products/jini/index.jsp

[20] http://www.research.ibm.com/journal/sj/373/wyckoff.html

[21] CSpace coordination model = “persistent publish, read and subscribe” + “semantics”

account that agents can write information in terms of the logical theories stored in their own individual CSpaces and not of the destination CSpaces. Thus, the coordination model has to be aware about this situation to request query rewriting and data transformation services.

Table 4. API for CSpaces coordination model

API call and description
Void write (set tuples, URI cs_destination, URI cs_origin) Write one or more tuples in a concrete CSpace (cs_destination) identified by a URI. If the tuples are defined in terms of a domain theory stored in a different CSpace then it is specify in the third parameter (cs_origin)
Tuple take (Template|Query t, URI cs_destination, URI cs_origin) Return the first tuple (or nothing) that match with the template or a query expressed in using a formal query language) and delete the matched tuple from a concrete CSpace cs_destination. If the template of query t is defined in terms of a different domain theory than the one stored in cs_destination, then it is specified in the parameter cs_origin
Tuple waitToTake (Template|Query t, URI cs_destination, URI cs_origin) Like take but the process is blocked until the a tuple is retrieved
Tuple read (Template|Query t, URI cs_destination, URI cs_origin) Like take but the tuple is not removed
Tuple waitToRead (Template|Query t, URI cs_destination, URI cs_origin) Like read but the process is blocked until a tuple is retrieved
Set scan (Template|Query t, URI cs_destination, URI cs_origin) Like read but returns all tuples that match with template or query t
Long countN (Template|Query t, URI cs_destination, URI cs_origin) Return the number of tuples that match template or query t
URI subscribe (URI agent, Template|Query t, Callback c, URI cs_destination, URI cs_origin) An agent (consumer) expresses its interested on tuples that match with template or query t in a concrete CSpace (cs_origin). Like take, if the template of query t is defined in terms of a different domain theory than the one stored in cs_destination, then it is specified in the parameter cs_origin. Any time that there is an update in the CSpace, the subscriber receives a notification that there are tuples available that match the template. The notification is executed by calling a method/routine specified by the subscriber. The operation returns an URI that identifies the subscription.
Set unsubscribe (URI agent, Template|Query t, Callback c, URI cs_destination, URI cs_origin) An agent (consumer) deletes its subscription, and no more related notifications are received. The operation returns a set of URIs of subscriptions deleted
URI advertise (URI agent, Template|Query t, URI cs_destination, URI cs_origin) An agent (producer) shows its intention to provide tuples that match t. Advertisement provides information to the system that can be used in advance to improve the distribution criteria of data and participants. The operation returns an URI that identifies the advertisement created.
Set unadvertise (URI agent, Template|Query t, URI cs_destination, URI cs_origin) An agent (producer) shows its intention to do not provide more tuples that match t. The related advertisements are deleted, and the operation returns a set of URIs deleted.
URI getTransaction (URI cs) Ask the CSpace infrastructure to create a new transaction and returns its id as a URI.
Boolean beginTransaction (URI txn, URI cs) Identify the beginning of a set of instructions executed under a concrete transaction (identified by a URI). Several processes can execute instructions under the same transaction, and only those processes can see the changes produced in the space before the transaction is committed.
Boolean commitTransaction (URI txn, URI cs) Make permanent a set of changes defined inside of a transaction txn.
Boolean rollbackTransaction (URI txn, URI cs) Undo a set of changes defined inside of a transaction txn.

In a Semantic Web mostly composed by machine processable semantics, the "persistent publish, read and subscribe" metaphor could be the common interaction model for applications and human users [Martin-Recuerda, 2005].

2.4.4 Semantic interoperability and consensus-making model

[Davenport and Prusak, 1998] mentioned that sharing knowledge requires a common language: "*People can't share knowledge if they don't speak a common language.*" The Semantic Web relies on ontologies for building this common language; however according to [Paslaru-Bontas, 2005], existing ontologies are at most formal specifications of conceptualizations, but they are rarely built to be *shared* and *reused*. CSpaces aims to bring ontologies again to their original purpose [Borst, 1997; and Gruber, 1993]). To achieve this goal, the following proposals are considered in the scope of CSpaces:

- ✓ Shared CSpaces are the places in which "*shared*" ontologies, rules and in some cases instances[22] are published.
- ✓ The generation of new Shared CSpaces will be driven by the interoperability necessities of users of the Semantic Web and will be constrained by the structure defined by the organizational model.
- ✓ Enforcing communication will be mainly done through Shared CSpaces using the set of operations that the coordination model provides (see previous section) and the ontological terms on which members of each Shared CSpaces have agreed.
- ✓ Shared CSpaces will become mediated repositories of schemas, rules and instances that will facilitate knowledge discovery and knowledge sharing.
- ✓ The construction of "*shared*" ontologies, rules and instances should follow a cooperative human-centered approach [Hoffman et al, 2002] in which knowledge workers are actively involved in ontology management tasks throughout the ontology engineering life cycle.

Although the study reported by [Visser and Cui, 1998; and Uschold, 2000] shows that it is difficult to create ontologies from a consensus process, most of the interoperability capabilities that the Semantic Web, Semantic Web Services and CSpaces promise, relies on the creation and use of ontologies as "shared" specification of conceptualizations. Also, [Cook and Brown, 1999; and Kotis and Vouros, 2003] suggest that a consensual process for the generation of merged ontologies not only is a means to exchange knowledge, but also a mean to generate new knowledge. In particular, [Kotis and Vouros, 2003] suggest that the creation of ontologies should follow a collaborative approach in which contributors should participate in structured conversations about conceptualizations. HCOME provides tool support for incorporating suggestions/positions to specifications that enable constructive criticism and avoids potential deadlocks.

[22] Instances sometimes can be part of private or sensitive information that companies or users would not like to make available. In those cases, instances would not be stored in Shared CSpaces but in individual CSpaces.

Argumentation is based on the construction of arguments and counter-arguments (defeaters) and the selection of the most acceptable of these arguments [Amgoud and Kaci, 2005]. One of the main advantages of computer supported argumentation is to structure persistent design discussions that can be checked later on.

The argumentation platform included in CSpaces is designed based on the following principles:

- ✓ Arguments are semantically specified based on an ontological argument model. Reasoning capabilities can be used to verify coherence and consistency of the argumentation, trace decisions and facilitate the generation of a single discourse graph.
- ✓ A discussion moderator (acting as a trusted-third-party) can participate in a discussion and guarantee an appropriate and fruitful end.
- ✓ A voting process is used to reach the goal of capturing the consensus on the generic arguments.
- ✓ A graphical visualization tool helps users to better understand each single discourse graph.

Editing operations in each Shared CSpace will be constrained with the dependencies that the affected terms have associated with them. CSpaces have associated metadata about themselves. The data of a CSpace can have dependencies (users and applications for each data element) explicitly stored. These dependencies can be generated by monitoring services that continuously analyze operations in each CSpace. Deletion and modification of data elements without dependencies will not require the initiation of an argumentation process. However, in the case that a member disagrees about a change, the versioning service will allow undoing writing operations executed on a Shared CSpace.

2.4.5 Security and trust model

The definition of a security model for CSpaces is challenging due to its open and decentralized nature. Unlike client-server systems in which certain nodes can be easily identified as a trustworthy server under certain premises, the nodes of the CSpace infrastructure may provide no such guarantee. Belong to a CSpace network does not mean that nodes are trustworthy to route queries, store machine processable semantics, or serve authentication credentials. [Gutierrez et al., 2004] identifies the following security issues that CSpaces should address:

- ✓ *Authentication*.
  Any application/user in an interaction may be required to provide authentication credentials by the other party. Identity-based authorization [Needham and Schroeder, 1978; Lampson et al., 1992; Kohl and Neuman, 1993] and capability-based systems [Gong, 1989; Bull et al., 1992; and Hayton, 1996] have been the typical means to achieve authentication requirements. In comparison with the identity-based approach, capability-style authorization is more suited for distributed systems, as it encourages distributed security management and is hence inherently more scalable.
- ✓ *Authorization*.

Applications should include mechanisms that allow them to control the access to the services being offered. Authorization in CSpaces should also guarantee anonymity of the agents to interact in the system. [Aberer et al., 2005] proposes a distributed public key infrastructure (PKI) in P2P infrastructures.

- ✓ *Confidentiality*.
  Keeping the information exchanged among nodes secret is another of the main properties that should be guaranteed in order to consider the channel secure. Confidentiality is achieved thanks to ciphering techniques.
- ✓ *Integrity*.
  This property guarantees that the information received by a party remains the same as the information that was sent from the client. Solutions to the integrity problem usually involve adding some type of redundancy to messages in the form of a "signature.". Techniques such as CRCs (cyclic redundancy checks), hashing, MACs (message authentication codes), or digital signatures (using symmetric or asymmetric encryption) are well-understood solutions to the integrity problem.
- ✓ *Non-repudiation*.
  It is necessary to be able to prove that a client used a resource from one provider (requester non-repudiation) and that the provider processed the client request (provider non-repudiation). This security issue is covered by means of digital signatures.
- ✓ *Availability***.**
  The need to take care of the availability aspects for preventing Denial of Service attacks or to arrange redundancy systems is also required. A Denial-of-service (DoS) attack attempts to make a node and its resources unavailable by overloading it. DoS attacks can be preventing by taking advantage of loosely constrained protocol features.
- ✓ *End-to-end security*: The CSpace infrastructure (also called *blue-storm*) will rely on a broad combination of light (mobile phones, PDAs, etc) and heavy devices (desktop computers, servers, etc). All of these systems rely on the ability for data processing intermediaries to receive and route data [IBM and Microsoft, 2002].

The openness and distributed model that CSpaces promotes consequently results in communication and collaboration with strangers. In truly open environments, the notion of trust is intimately engaged with security. Several authors claim that *trust and security are two sides of the same coin.*

Definition 1 (Trust) [Grandison and Sloman, 2000] *the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context*.

Definition 2 (Trust Relationship) [AbdulRahman, 2005] *A trust relationship exists when an entity has an opinion about another entity's trustworthiness. Thus, trust relationships do not exist between strangers or an entity that has no knowledge about another's existence and as consequence, trust relationship is not transitive.*

Trust management can be classified into three categories: credential and policy-based trust management, reputation-based trust management, and social network based trust

management. According with [Suryanarayana and Richard Taylor, 2004], the three approaches can be complementary[23].

The primary goal of **credential and policy-based trust management systems** is to enable access control. In such systems, [Blaze et al., 1996; Kagal et al., 2001; Yu, et al., 2001; Li et al., 2002; and Yao, 2003], peers use credential verification to establish a trust relationship with other peers.

Reputation-based trust management systems on the other hand focus on trust computation models capable to estimate the degree of trust that can be invested in a certain party based on the history of its past behavior. The main issues of such systems are how to model and compute trust, and how to manage reputation data. Several proposals have been described in the literature including SPORAS and HISTOS [Zacharia and Maes, 2000; Zacharia and Maes, 1999], XREP [Damiani et al., 2002], NICE [Lee et al., 2003], DCRC/CORC [Gupta et al., 2003], Beta [Josang and Ismail, 2002], EigenTrust [Kamvar et al., 2003], etc.

The third kind of trust management systems, in addition, utilizes social relationships between peers when computing trust and reputation values. Regret [Sabater and Sierra, 2001] that identifies groups using the social network, and NodeRanking [Pujol et al., 2002] that identifies experts using the social network are examples of **social network based trust management systems**.

One input that is often used in a trust decision making process is the reputation of the partner. Reputation is a powerful distributed mechanism of social control that has the potential to purge society of 'bad' agents and promote 'good' ones.

Definition 3 (Reputation) *Reputation can be regarded as a unitary appreciation of the personal attributes of the truster: competence, benevolence, integrity and predictability.*

Definition 4 (Trustworthiness) [AbdulRahman, 2005] *An agent's trustworthiness is his reputation for being worthy of a certain level of trust in a given situation.*

The *trust model* proposed for CSpaces relies in the following principles:

- ✓ The CSpaces model will incorporate features from three different and complementary models: **credential and policy-based** trust management, **reputation-based** trust management, and **social-network-based** trust management.
- ✓ **Agents** (active entities human or not) that access or own a CSpace are associated with a *unique ID*.
- ✓ In CSpaces a **trust relationship** exists between two agents when one agent has an opinion about the other agent's trustworthiness. A trust relationship is binary (only between two agents), unidirectional and dynamic (may change over the time.

[23] [Bonatti et al., 2005] support that policy-based and trust management can be complementary and can enhance the properties of the existing trust management tools.

- ✓ An **opinion** indicates a subjective agent's belief about another's trustworthiness, and it is measured based on a defined set of trust levels. Opinions are collected by a reputation mechanism associated with each CSpace and stored as a part of their metadata description. The opinion of an agent about another agent can differ between two different CSpaces in which both agents are members.
- ✓ An agent can have several reputation scores, each of them associated with a different CSpaces in which the agent is member. A **local reputation score** is calculated from the opinions of the rest of the members of a CSpace. A member of a CSpace cannot access the opinions of other members about himself, but can ask about his local reputation score.
- ✓ A **global reputation score** is calculated from local scores and stored in the Shared CSpaces that also maintain a general information catalog of the rest of CSpaces. The calculation of the general reputation scored by an agent is influenced by the number of members of each CSpace that provide local reputation scores and the amount of dependencies that each CSpace has with other CSpaces (similar to the PageRank algorithm including transitive dependencies).

2.4.6 Knowledge access model

Knowledge access comprises all of the techniques and mechanisms that facilitate the exploration, visualization and editing of semantic formal representations of information stored in knowledge bases. Knowledge access aims to improve the creation, comprehension and transfer of knowledge by exploiting graphical and natural language representation means.

Graphical representation of knowledge was intensively studied in the previous decades and is still ongoing research (please refer [Eppler and Burkard, 2004] for a survey). The popularization in the use of ontologies brings into focus the necessity to provide graphical visualization as an essential feature for ontology editing and browsing tools. Tree and graph visualization approaches are the more common techniques to graphically represent ontologies. A concrete solution for displaying large tree structures, called *hyperbolic tree* [Lamping et al., 1995], was developed in 1995 in Xerox Parc Laboratories and commercialized by Inxight[24] . This technique is used in tools like KAON[25] and KIM[26]. Graphical representation facilities should be available in user interfaces for editing and browsing the content of a CSpace or a set of them.

A complementary approach for knowledge access in which semantic data descriptions are presented in a user friendly way is natural language generation (NLG). "*NLG takes structured data in a knowledge base as an input and produces Natural Language text, tailored to the presentational and the target reader*" [Reiter and Dale, 2000]. NLG mechanisms can constantly keep up-to-date text descriptions of data semantics and can

[24] http://www.inxight.com
[25] http://kaon.semanticweb.org/
[26] http://dell.sirma.bg/kim/graph/Graph.jsp

automatically provide those text descriptions in multiple languages [Bontcheva, 2004]. Current efforts in NLG have two main foci. The first one is to provide tools specific oriented to semantic web platforms, and the second one is to design NLG systems that keep the system simple enough to be maintained by non-NLG experts, but without losing quality of the text output ([Bontcheva and Wilks, 2004; Wilcock, 2003; Wilcock and Jokinen, 2003; and Bontcheva, 2005]).

To facilitate the understanding of the information showed by the graphical interface, CSpaces will extend the technology developed in ONTOSUM [Bontcheva, 2005], a generator for textual tailored summaries from ontologies. ONTOSUM is based on a well tested technology [Bontcheva and Wilks, 2004], it is domain-independent, it is designed for non-NLG experts, and it supports entries in different formal ontology languages like RDF(S), DAML+OIL and OWL.

ONTOSUM is implemented as a pipeline system [Reiter and Dale, 2000] inside of the GATE infrastructure [Bontcheva et al., 2003]. Althought the integration with GATE reports a lot of benefits, in CSpaces would be interesting to disaggregate the NLG components and build an independent tool that can be executed in light-weight devices. The generator, HYLITE+, is implemented in Prolog and can run in diverse platforms.

Finally, *Controlled Natural Languages*[27](CNL) are subsets of natural languages whose grammars and dictionaries have been restricted in order to reduce or eliminate both ambiguity and complexity. In the context of CSpaces, CNL will facilitate non-logician users to edit the information (mostly domain theory and instances) stored in a CSpace.

2.4.7 Architecture model (*blue-storm*)

Given that CSpaces aims to re-elaborate the Semantic Web proposal by minimizing syntactic data representation, many of the design considerations for the Semantic Web architecture are still valid for CSpaces [Martin-Recuerda, 2005]. Scalability, distribution and decentralization are three requirements that CSpace and Semantic Web architectures have in common. However, the CSpace coordination model built on the "persistent publish, read and subscribe" metaphor requires an architecture model that can deal with asynchronous communication. A second difference in the two infrastructures is the organization of metadata around Individual and Shared CSpaces.

Like the Semantic Web, a potential first approach is to build CSpaces upon the existing Web infrastructure that has been described using an abstract model called REST (Representational State Transfer) [Fielding, 2000]. The fundamental principle of REST is that resources are stateless and identified by URIs. HTTP is the protocol used to access to the resources and provides a minimal set of operations enough to model any applications domain [Fielding, 2000]. Those operations (GET, DELETE, POST and PUT) parallel closely Tuple-Space operations (READ, TAKE and WRITE in TSpaces)[28]. Tuples can be identified by URIs and/or can be modeled using RDF triples (as [Fensel, 2004] suggests). Since every representation transfer must be initiated by the client, and every response

[27] http://www.ics.mq.edu.au/~rolfs/controlled-natural-languages/
[28] There is a brief discussion of HTTP and Linda at http://rest.blueoxen.net/cgi-bin/wiki.pl?LindaAndTheWeb

must be generated as soon as possible (the statelessness requirement) there is no way for a server to transmit any information to a client asynchronously in REST. Furthermore, there is no direct way to model a peer-to-peer relationship [Khare and Taylor, 2004]. Several extensions of REST, like ARRESTED [Khare and Taylor, 2004], have been proposed to provide a proper support of decentralized and distributed asynchronous event-based Web systems.

The limitations of REST to model asynchronous interaction motivated that Martin-Recuerda pays attention to Peer-to Peer systems. They are decentralized, distributed, self-organized and capable of adapting to changes such as failure [Pietzuch, 2004]. Although there are several open issues regarding scalability, shared resources management, security and trust [Bawa et al., 2003], current efforts in the field (for instance, [Rhea et al., 2003; and Aberer et al., 2003]) are progressively overcoming these problems.

The preliminary proposal for CSpaces architecture, outlined in this section, is strongly influenced by the work done in OceanStore[29], Edutella[30] and SWAP[31]. Three kinds of nodes are identified in CSpaces architecture: CSpace-servers, CSpace-heavy-clients and CSpace-light-clients.

- ✓ **CSpace-servers** store primary and secondary replicas of the data published in individual and shared CSpaces; support versioning services; provide an access point for CSpace clients to the peer network; maintain and execute reasoning services for evaluating complex queries; implement subscription mechanisms related with the contents stored; provide security and trust services; balance workload and monitor requests from other nodes and subscriptions and advertisements from publishers and consumers.
- ✓ **CSpace-heavy-clients** provide a storage infrastructure and reasoning support to let users to work off-line with their own individual and shared spaces. Replication mechanisms are in charge to keep replicas in clients and servers up-to date. In addition, these clients also include a presentation service (based on NLG and Knowledge visualization techniques) to facilitate the visualization and edition of knowledge contents.
- ✓ **CSpace-light-clients** only include the presentation infrastructure to write query-edit operations and visualize knowledge contents stored on CSpace-servers.

When clients are online and connected with the rest of the nodes of the system through an access point (server node) they have the obligation to share computational resources (CPU time, memory and persistent storage services). Thus CSpace-servers can divert client's resources demanding requests, and consequently, alleviate temporarily the workload of servers. If the client is a heavy-client, requests that can be performed locally will not be sent to CSpace-servers. Periodically, replicas will be updated to keep heavy-clients and servers up-to-date.

[29] http://oceanstore.cs.berkeley.edu/
[30] http://edutella.jxta.org/
[31] http://swap.semanticweb.org/public/index.htm

Commonly those hybrid architectures that combine pure P2P and client/server systems are called **super-peer** systems. CSpace-servers are formally peers, but it is not the case of CSpace-clients that promote a client-server relation with CSpace-servers. Like OceanStore [Rhea et al., 2003], this configuration drives into two-tiered system. The upper-tier is composed of well-connected and powerful servers, and the lower-tier, in contrast, consists in clients with limited computational resources that are temporarily available.

It is expected that the CSpaces infrastructure will be self-organized as in other peer-to-peer systems and will include monitoring mechanisms that will analyze the distribution of the data in the different nodes and the data flows between these nodes. Data stored in server's and client's access points will be re-distributed in appropriate configurations that minimize the network traffic and maximize the semantic similarity of the data stored in the closer peers. Subscriptions and advertisements from publishers and consumers will provide useful information to determine optimal configurations where consumers and publishers with common interests will be connected to closer servers. In addition, the definition of Shared CSpaces will be other information source to determine semantic similarity between nodes.

The communication metaphor will differ from most of the P2P implementations that use message passing. Just as OceanStore is built on top of an event-based architecture[32], CSpace promotes the coordination model "publish, read and subscribe" for the communication of its nodes. In addition, the use of topologies that simulate spanning trees (e.g. HyperCup in Edutella) will reduce unnecessary data flows and will facilitate the implementation of replication mechanisms.

Together with the peer infrastructure, a set of registered agents (software applications and human users) will play the roles of producers and consumers of information through a "publish, read and subscribe" coordination mechanism. Those agents will take advantage of a group of management services that *blue-storm* will provide in order to:

- ✓ Facilitate the visualization and comprehension of the information stored. A detailed description is provided in section 2.4.6
- ✓ Provide distributed reasoning services that are able to return meaningful answers in the presence of inconsistency between the content stored in difference and related CSpaces.
- ✓ Provide transaction support for a group of write/read operations executed by multiple agents. Rollback mechanisms will allow undoing all the operations executed during a transaction and return to a state in the CSpace in which those operations were not executed.
- ✓ Publish and subscribe services according to the proposal of section 2.4.3.
- ✓ Allow members of the system to generate Shared CSpaces through a collaborative process supported by an argumentation mechanism.
- ✓ Provide a versioning infrastructure that includes tracking changes and diff tools.

[32] More precisely Pond, the OceanStore prototype, which is built on top of an event-based system

- ✓ Store metadata related with the dependencies between applications-users and each element of a CSpace. Elements (concepts, rules and instances) with dependencies will be more difficult to delete/modify.
- ✓ Analyze and store the activity of the users and applications that are interacting through a concrete CSpace using monitor services. The information collected by those services can be used to
  - → Identify dependencies between applications/users and the data stored, and plan redistribution of the information between peers that can maximize the performance of the entire system.
  - → Restrict the ability of users and applications to perform delete and modify operations over tuples that have dependencies with other applications/users

2.5 Summary

In this section we summarize with a table (on the following page) the proposals for tuplespace-based computing in the Semantic Web in terms of some fundamental building blocks identified for such systems.

	Semantic Data Model	Organizational Model	Co-ordination Model	Collaborative and Consensual Model	Architecture Model	Security and Trust Model
STuples	ServiceTuple and DataTuple contain a DAML+OIL Instance	Centralised space	JavaSpace model extended by Publish-Subscribe	Not defined	JavaSpaces	Not defined
Triple Space Computing	RDF Triples	Multiple independent triple spaces	TSpace model extended to handle multiple read and write operations, Publish-Subscribe	Not defined	REST	Not defined
Semantic Web Spaces	RDFTuples with the structure <s,p,o,id> Data (RDF Syntax) and information (RDF Semantics) views on the space	*Contexts* based on idea of scopes	Classical Linda model extended with dedicated primitives for RDFTuple interaction at the syntactic and semantic level	Mediation between content and semantics using dedicated matching algorithms and components	Self-organized, distributed architecture Ontological description of the space	Access and trust policies referenced in model of the space Reference monitor (dedicated security component)
CSpaces	Tuple <guid, fm, type, sguid, vguid, mguid> *fm* contains a formal logic language such as RDF Specification of relations between tuplespaces using mappings	Individual and Shared CSpaces DAG configuration of interconnected CSpaces	TSpace model extended to handle multiple read and write operations, Publish-Subscribe, transactions	Argumentation mechanism for the collaborative creation of Shared CSpaces Shared CSpaces are mediation spaces for data and process integration	Super peer (P2P) Metadata associated with each space Distinction between “raw” and “reasoning” spaces	Security and trust data is maintained in each semantic tuplespace. Credential and policy-based trust management, reputation-based trust management, and social-network-based trust management.

3 Towards a Unified Conceptual Framework

In this section we discuss the previous proposals for tuplespace-based computing in the Semantic Web (sTuples, Triple Space Computing, Semantic Web Spaces and CSpaces). In order to guide us in the determination of a *unified conceptual framework* from these proposals let us compare them in terms of each of the models introduced here.

3.1 Semantic and data model

The lowest common denominator of the aforementioned approaches to semantics-enabled tuplespace computing w.r.t. the underlying semantic model is the support of new tuple types storing data expressed in a particular Semantic Web representation language. Excerpt for the sTuples proposal, the remaining three tuplespace approaches foresee a certain level of support for RDF data. Semantic Web Spaces and CSpaces provide first ideas w.r.t. RDFS, OWL and SWRL support. Usually tuples are extended with an identification mechanism. CSpaces and Semantic Web Spaces also foresee a means to (optionally) attach provenance information to individual (or sets of) tuples; however, they resort to slightly different interpretations of the provenance concept. Further on, CSpaces introduce versioning information to the classical tuple notion, and the unique id of the creator of the tuple for trust purposes. On the other hand, versioning is out of the central focus of Semantic Web Spaces and sTuples.

As a conclusion, CSpaces presents the richer notion of data-model which includes all requirements specified by sTuples, Triple Space Computing and Semantic Web Spaces. The latter together with CSpaces define an upper level layer on top of the data layer. In the case of Semantic Web Spaces this is called *information view* and visualizes tuples as RDF graphs. In the case of CSpaces, tuples are logically grouped in knowledge containers that store a logical theory, the relations (mappings) with other logical theories (other CSpaces), relations with real world objects (annotations), security and trust information, and a metadata characterization of the CSpace itself. This metadata should be ontologically described in Semantic Web Spaces and CSpaces. However, only Semantic Web Spaces provides currently a specification of such an ontology.

3.2 Organizational model

The organizational model is explicitly taken into consideration in three of the presented approaches (Triple Space Computing, Semantic Web Spaces and CSpaces). Triple Space computing [Bussler, 2005] and CSpaces explicitly include the notion of several independent tuplespaces, although in the case of CSpaces, the CSpaces which have domain dependencies are interconnected by mapping rules. Semantic Web Spaces, on the other hand, only mentions one global space that can be partitioned using the notion of contexts.

A separate issue that can strongly influence the organizational model is how to handle heterogeneity in the data that is exchanged in a space. sTuples does not include any proposal for this problem and assumes that all participants who publish in a sTuple space implicitly agreed on a common vocabulary specification. Triple Space Computing and Semantic Web Spaces rely on mediation services/components that express the mappings between heterogeneous sources. In particular, Triple Space is part of WTriple (i.e.

WSMO/L/X + Triple Space) which is the technical kernel of Semantically Empowered Service-Oriented Architectures (SESA, [Fensel, 2005]). Mediation services will be provided by the WTriple semantic execution environment (i.e. WSMX [Zaremba and Moran, 2005]). However, it is not well defined how mediation services will be used in Triple Space and Semantic Web Spaces. As an opposite approach, CSpaces integrate mediation capabilities as a part of its infrastructure. Shared CSpaces become mediated persistent communication channels interconnected with other CSpaces through mapping rules. Interoperability requirements will drive the generation of new Shared CSpaces that ideally will be organized as a DAG model of interconnected spaces.

3.3 Coordination model

The coordination model underlying the four approaches is Linda with some extensions. In addition, all proposals take into account the advantage of Semantic Web technologies to improve the matching abilities of retrieval operations. However, only Semantic Web Spaces distinguish between operations at data level and semantic level. The former group of operations guarantees backward compatibility with classical Linda applications.

On the other hand, sTuples, CSpaces, and the version of the Triple Space described in [Martin-Recuerda and B. Sapkota, 2005] extend tuplespace coordination model with publish-subscribe capabilities. This extension, already considered in commercial implementations of tuplespace computing like TSpaces and JavaSpaces, overcomes one of the major limitations of tuplespace computing: the flow-coupling of consumer applications. In addition, the combination of tuplespace computing and publish-subscribe improves the latter avoiding the *event-storm* problem[33].

Finally, the unified coordination API will be based on the Semantic Web Spaces API, Triple Space API [Martin-Recuerda and B. Sapkota, 2005] and CSpaces API, but it is still under discussion, and it will be included in the next revision of this technical report.

3.4 Collaborative and consensus-making model

The only approach which pays special attention to collaborative aspects is CSpaces. The remaining tuplespace proposals do not deal with this issue in detail, mainly because they do not consider collaboration and consensus as a core focus of their initial semantic tuplespace infrastructure. The authors of sTuples, Triple Space Computing and Semantic Web Spaces do not yet propose how components will provide and store the required ontologies and will take care of their maintenance. The same happens with the mediation services that in sTuples are not even considered and in Triple Space and Semantic Web Spaces are vaguely described as additional components. One of the key proposals in CSpaces is their use not only as a persistent and asynchronous communication channel, but also as a knowledge container. Thus, message content can refer to ontological terms stored (or referenced) in a concrete CSpace, and messages can expand the knowledge

[33] Event storms is one of the most important scalability issues to have been reported in publish-subscribe systems, and are produced by a large number of concurrent notifications that usually also have attached large data sets [Fielding, 2000].

base adding new information to a CSpace. Martin-Recuerda also believes that it is necessary to recover the idea of ontologies as "*shared*" conceptualizations by the members of a CSpace. Thus, the integration of consensual-making tools is highly recommended by the CSpace proposal.
To conclude the discussion of this sub-section, it is fair to mention that CSpaces is not only targeted at realizing a Semantic Web-enabled coordination middleware (the case of the other approaches), but also sees coordination technologies as a means to enable distributed knowledge sharing on the Semantic Web.

3.5 Security and trust model

The necessity of a security and trust component for semantics-enabled tuplespaces is well-recognized by Triple Space Computing (according to [Bussler, 2005]), CSpaces and Semantic Web Spaces. The two latter approaches foresee a dedicated component supporting credentials, policy-based, reputation-based and trust management features. Furthemore, CSpaces supports social-network trust management capabilities based on peer relations, and CSpaces also includes in the data model specification the id of the creator that can be used for filtering tuples written by agents with low reputation, and security-trust information is stored as a part of each CSpaces. From a security point of view, CSpaces identifies most of the security issues that a unified proposal should address: authentication, authorization, confidentiality, integrity, non-reputation, availability, and end-to-end security.

3.6 Architecture model

The architecture of the envisioned systems mainly relies on distributed and decentralized models. While sTuples resort to the architecture underlying JavaSpaces, the remaining approaches foresee an architectural model supporting decentralization, while converging in terms of requirements like scalability. Triple Space Computing (according to [Bussler, 2005]) follows REST principles, and CSpaces induce decentralization and self-organization by means of peer-to-peer ideas, while Semantic Web Spaces aims at tackling these issues using intelligent distribution strategies (e.g. self-organization on swarm intelligence principles). According to [Martin-Recuerda, 2005], REST is not the appropriate solution if the tuplespace coordination model is extended with notification capabilities. Thus, a super-peer approach together with the implementation of intelligent self-organization mechanisms (for instance, swarm intelligence) will be considered in the next revision of this document.

3.7 Summary

In this section we summarize the current status of the unified conceptual framework for semantic-enabled tuplespace computing.

Conceptual and Architecture Model	Unified Proposal
Semantic Data Model	**<guid, fm, type, sguid, vguid, mguid>** ✓ *fm* can be defined using a RDF triple (i.e. <s, p, o>) or a formal logic language that provides ontological modeling primitives and rule support. ✓ Distinction between syntax and semantics-oriented data management. ✓ Security and trust data is maintained in each semantic tuplespace[34]. ✓ Specification of relations between semantic tuplespaces. ✓ Associated metadata to spaces described using an ontology
Organizational Model	✓ Virtual and physical space partition management features. ✓ Contexts will virtually partition a concrete semantic tuplespace. ✓ Specifications of conceptualization agreements will drive the creation of semantic tuplespaces ✓ Ideally, the organization model should follow a DAG configuration of interconnected semantic tuplespaces (e.g. tree structure as in CO4 [Euzenat, 1995])
Co-ordination Model	✓ Linda coordination model with extensions: ✓ Extensions to handle multiple read and write operations ✓ Extensions to provide notification using publish-subscribe approaches. ✓ Extensions for transaction support
Collaborative and Consensus-Making Model	Under discussion
Security and Trust Model	✓ Credential and policy-based trust management, reputation-based trust management, and social-network- based trust management. ✓ Security and trust management should be maintained using a decentralized configuration. Thus, each semantic tuplespace should maintain information related to its security and trust data.
Architecture Model	✓ Super-peer model for storing semantic tuplespaces. ✓ Semantic tuplespaces can be stored in one node or in several nodes.

[34] Associating security-trust information to each space is a way to keep the architecture decentralized.

Conceptual and Architecture Model	Unified Proposal
	✓ A node can stored one or more semantic tuplespaces ✓ Ontological description of the conceptual model ✓ The architecture model should provide the following services: – Provide distributed reasoning services that are able to return meaningful answers in the presence of inconsistency – Provide transaction support for a group of write/read operations executed by multiple agents. – Management of subscriptions and notifications. – Versioning infrastructure that includes tracking changes and diff tools. – Analysis and storage of the activity of the users and applications that are interacting through a concrete semantic tuplespace.

4 Applying semantic tuplespaces paradigm to Semantic Web Services

Web Services have added a new level of functionality to the current Web, making the first step to achieve seamless integration of distributed components. However, current proposals of Web Services have two major drawbacks:

- ✓ Web Services only address the syntactical aspects of a Web Service and, therefore, only provide a set of rigid services that cannot adapt to a changing environment without a high degree of human intervention [Fensel and Bussler, 2002].
- ✓ Web Services are based on the message-exchange paradigm, and thus, they are not fully compliant with core paradigms of the Web itself. Moreover, message exchange paradigm requires from Web Services a strong coupling in terms of reference and time [Fensel, 2004].

Semantic Web Services try to tackle the former issue by using explicit, machine-understandable semantics, in order to improve the degree of automation in locating, combining and using of Web Services. The Web Service Modeling Ontology (WSMO) [Roman et al., 2005] is one of the most promising proposals for describing all relevant aspects related to general services which are accessible through a web service interface. WSMO has its conceptual basis in the Web Service Modeling Framework (WSMF) [Fensel and Bussler, 2002], refining and extending this framework and developing a formal ontology (WSMO) and set of languages tailored for modeling Web Services (WSML).

WSMO provides a unifying view of a service; the value the service can provide is captured by its capability, and the means to interact with the service provider to request the actual performance of the service, or to negotiate some aspects of its provision, is captured by the service interfaces. The software entity able to provide the service is transparent to us, and we are only concerned with its interaction style and with what other services are used to actually provide the value described in the capability. A service description consists of one capability, which describes the functional aspects of a service, non-functional properties, and one or more interfaces [Roman et al., 2005]. An interface describes the choreography and the orchestration of the service. The choreography specifies how the service achieves its capability by means of interactions with its client - i.e. the communication with a client of the service; the orchestration specifies how the service achieves its capability by making use of other services - i.e. the coordination of other services. Figure 5 shows the core elements that are part of the description of a WSMO service.

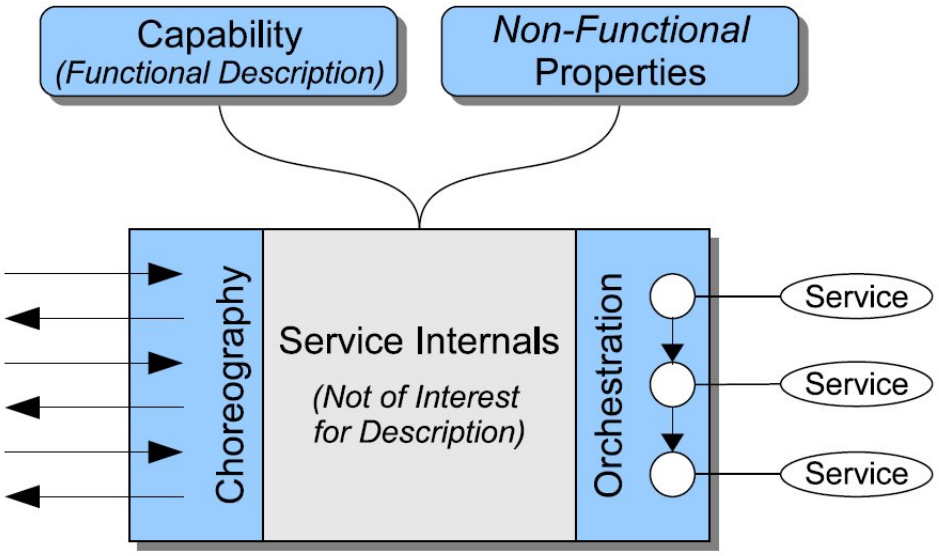

Figure 5: WSMO service description overview [Roman et. al., 2005].

The interaction with a service described by WSMO can be done using WSDL [Christensen et. al., 2001] and SOAP. Instead of doing this, we propose to ground services on top of semantic tuplespace computing paradigm. Given that the API for the unified framework identified in previous section has not yet defined, we will provide an initial proposal for grounding WSMO choreography on top of CSpaces.

4.1 Interfaces in WSMO

An interface describes how the functionality of the service can be achieved (i.e. how the capability of a service can be fulfilled) by providing a twofold view on the operational competence of the service [Roman et al., 2005]:

- ✓ **choreography** decomposes a capability in terms of interaction with the service (from the client perspective).
- ✓ **orchestration** decomposes a capability in terms of functionality required from other services.

This distinction reflects the difference between communication and cooperation. The choreography defines how to communicate with the service in order to consume its functionality. The orchestration defines how the overall functionality is achieved by the cooperation of more elementary service providers.

The web service interface is meant primarily for behavioral description purposes of web services and is presented in a way that is suitable for software agents to determine the behavior of the service and reason about it; it might be also useful for discovery and selection purposes and in this description the connection to some existing web services

specifications e.g. WSDL [Christensen et. al., 2001] could also be specified. The definition of an interface is given below [Roman et al., 2005]:

```
Class interface
      hasNonFunctionalProperty type nonFunctionalProperty
      importsOntology type ontology
      usesMediator type ooMediator
      hasChoreography type choreography
      hasOrchestration type orchestration
```

Listing 4.1: Interface Definition[35]

4.1.1 Choreography

WSMO Choreography deals with interactions of the Web service from the client's perspective. We base the description of the behavior of a single service exposed to its client on the basic ASM model [Gurevich, 1995]. WSMO Choreography interface descriptions inherit the core principles of such kind of ASMs, which summarized, are: (1) they are state-based, (2) they represents a state by a signature, and (3) it models state changes by transition rules that change the values of functions and relations defined by the signature of the algebra.

In order to define the signature we use a WSMO ontology, i.e. definitions of concepts, their attributes, relations and axioms over these. Instead of dynamic changes of function values as represented by dynamic functions in ASMs we allow the dynamic modification of instances and attribute values in the state ontology.

Taking the ASMs methodology as a starting point, a WSMO choreography is state-based and consists of three elements which are defined as follows [Scicluna et. al., 2006]:

```
Class choreography
      hasNonFunctionalProperties type nonFunctionalProperties
      hasStateSignature type stateSignature
      hasTransitionRules type transitionRules
```

Listing 4.2: Choreography Interface

Non Functional properties

The non-functional properties of a service are aspects of the service that are not directly related to its functionality; apart of Dublin Core metadata set[36], specific elements for web services like Accuracy (the error rate generated by the service), Financial (the cost-related and charging-related proper- ties of a service [O'Sullivan et. al., 2002]), Network-related QoS (QoS mechanisms operating in the transport network which are independent of the service), Owner (the person or organization to which the service belongs), Performance (how fast a service request can be completed), Reliability (the ability of a service to

[35] WSMO is described using MOF metamodel facility (http://www.omg.org/mof/)
[36] http://dublincore.org/

perform its functions, i.e. to maintain its service quality), Robustness (the ability of the service to function correctly in the presence of incomplete or invalid inputs), Scalability (the ability of the service to process more requests in a certain time interval), Security (the ability of a service to provide authentication, authorization, confidentiality, traceability/audit-ability, data encryption, and non-repudiation), Transactional (the transactional properties of the service), Trust (the trust worthiness of the service), or Version.

State Signature

The signature of the machine is defined by (1) importing an ontology (possibly more than one) which defines the state signature over which the transition rules are executed, (2) an optional set of OO-Mediators if the imported state ontologies are heterogenous (3) a set of statements defining the modes of the concepts and relations and (4) a set of update functions. The types of modes that a concept or relation can be assigned are as follows:

- ✓ ***in*** - meaning that the extension of the concept or relation can only be changed by the environment. A **grounding** mechanism for this item may be provided that implements *write* access for the environment.
- ✓ ***out*** - meaning that the extension of the concept or relation can only be changed by the choreography execution. A **grounding** mechanism for this item must be provided that implements *read* access for the environment.
- ✓ ***shared*** - meaning that the extension of the concept or relation can be changed by the choreography execution and the environment. A **grounding** mechanism for this item may be provided that implements *read/write* access for the environment and the service.
- ✓ ***static*** - meaning that the extension of the concept cannot be changed. This is the default for all concepts and relations imported by the signature of the choreography.
- ✓ ***controlled*** - meaning that the extension of the concept is changed only by a choreography execution.

The default mode for concepts of the imported ontologies not listed explicitly in the modes statements is *static*. The modes are grounding by means of a URI reference to the document which describes such grounding. However, only in, out and shared modes are allowed to be grounded.

```
Class stateSignature
      hasNonFunctionalProperties type nonFunctionalProperties
      importsOntology type ontology
      usesMediator type ooMediator
      hasIn type mode
      hasOut type mode
      hasShared type mode
      hasStatic type mode
      hasControlled type mode
```

```
Class mode sub-Class {concept,relation}
      hasGrounding type grounding
```

Listing 4.3: Definition of the State Signature

Transition Rules

The most basic form of rules deal with basic operations on instance data, such as adding, removing and updating instances to the signature ontology. To this end, we define the atomic update functions to add and delete, as well as a update instances, which allow us to add and remove instances to/from concepts and relations and add and remove attribute values for particular instances:

- ✓ **add**(*fact*)
- ✓ **delete**(*fact*)
- ✓ **update**($fact^{new}$)
- ✓ **update**($fact^{old}$ => $fact^{new}$)

More complex transition rules are defined recursively, analogous to classical ASMs by **if-then**, **forall** and **choose** rules:

- ✓ **if** *Condition* **then** *Rules* **endIf**
- ✓ **forAll** *Variables* **with** *Condition* **do** *Rules* **endForAll**
- ✓ **choose** *Variables* **with** *Condition* **do** *Rules* **endChoose**

4.1.2 Orchestration

Describes how the service makes use of other services in order to achieve its capability.
In many real scenarios a service is provided by using and interacting with services provided by other applications or businesses. For example, the booking of a trip might involve the use of another service for validating the credit card and charging it with the correspondent amount and the user of the booking service may want to know with which other business organizations he is implicitly going to deal with.
WSMO introduces the orchestration element in the description of a service to reflect such dependencies. WSMO orchestration allows the use of statically or dynamically selected services. In the former case, a concrete service will be selected at design time. In the latter case, the service will only describe the goal that has to be fulfilled in order to provide its service. This goal will be used to select at run-time an available service fulfilling it (i.e. the service user could influence this choice). This aspect is still an ongoing work within the WSMO working group and thus we limit ourselves to consider choreography interfaces for the sake of grounding to CSpaces.

4.2 Semantic Web Services grounding for CSpaces

The model for describing choreography interfaces in WSMO abstracts away from the underlying protocol details which are used as a means of communication between a Web

service and the entity communicating with the service (the latter being automated or human). This is also thanks to the underlying ASM model which allows describing systems in an abstract way.

Currently, the primary grounding proposed within the WSMO community is based on WSDL as described in [Kopecky et al., 2005] but leaves the possibility to define other types of grounding. We present here a novel approach which describes how Semantic Web Services (based on WSMO) can be grounded to CSpaces. The CSpaces scenario provides a richer set of operations than WSMO Choreography descriptions and we thus present a partial solution which can serve as the basis for such grounding.

As described earlier, WSMO Choreography interfaces ground concepts and relations directly to WSDL messages in input operations. There are two aspects which should be considered when grounding semantic web service descriptions, namely, data grounding and grounding to operations. The former deals with the transformation of the semantic data to the message format handled by the underlying protocol which in the case of WSDL, the messaging format would be XML. The other aspect deals with using the appropriate operations of the underlying protocol to receive/send the data needed. For WSDL, this would imply a mapping to the underlying WSDL operations of the service. Similarly for grounding to CSpaces, two basic pieces of information are needed: a way to map to the operation and a way to bind a particular concept to a specific parameter of the operation. To this extent, we developed an ontology which is to be used to encode the information needed to specify the grounding (see Annex I at the end of this document).

Since WSMO Choreography is based on the ASM methodology, we clarify here how we envision such methodology would "map" to the CSpaces scenario. The space itself is somehow "detached" from the web services which are reading and writing to it. This is due to the fact that the different agents may have different signatures (that is, heterogeneous signatures are allowed). In terms of ASMs, an agent (a web service in our scenario) views the environment as the tuple space itself and the rest of the agents. However, there is no direct communication between the agents themselves since this happens through the CSpace. Tuples in the space are regarded as locations in ASMs, inheriting the classification properties of locations. In simpler terms, a tuple of type **static** cannot be updated by the agent who owns it and neither by the environment. A tuple of type **in** can only be updated by the environment and read by the agent. A tuple of type **out** can only be updated by the agent and read by the environment. A tuple of type **shared** can be updated and read by both the environment and the agent. Finally, a tuple of type **controlled** can only be updated by the agent but may also be read by the environment. Typically for CSpaces, it is more natural to define the tuples as *shared*. However, there might be cases where restrictions would apply on the tuples in the space and we thus leave the modeler of the choreography to define this. Note also that in terms of WSMO, the tuples described here correspond to the concepts and relations defined by some WSMO ontology and used by the choreography in some WSMO Web Service description. The data that it is exchanged by web services through WSMO choreography interfaces is instance data (for instance, "*James memberOf DERIEmployee*" where

DERIEmployee is a concept). A concept (or relation) can have a different type in each state signature (it can be an input variable for a web service and an output variable for another service). Thus, security policies are required to restrict the access of each web service to each instance according to the typed defined in its state signature for the associated concept-relation. Moreover, each Web Service can use a different ontology to describe terms in its state signatures. So heterogeneity should be taken into account and solved using Shared CSpaces or using external mediator services.

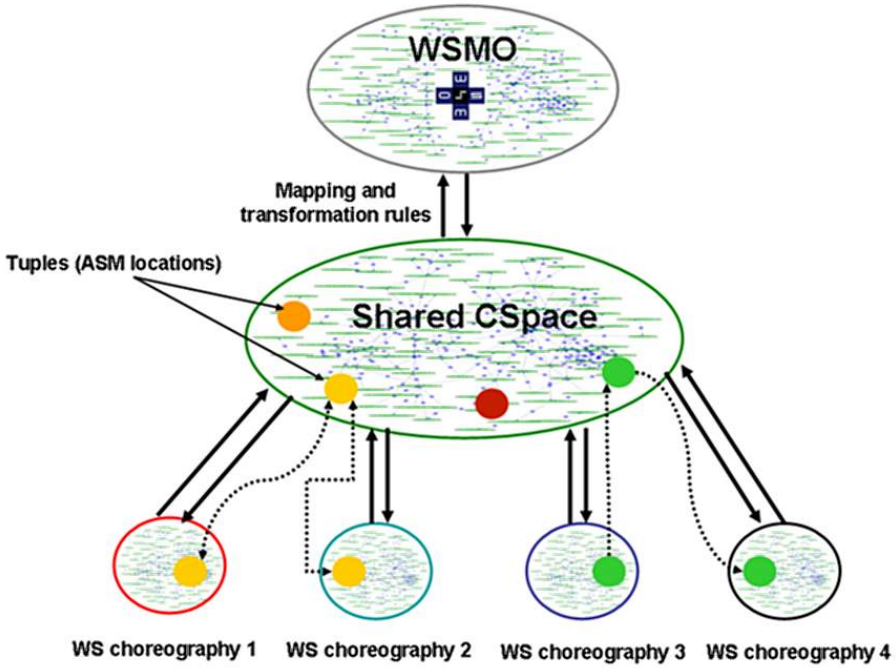

Figure 6 - CSpace and ASMs

CSpaces define a set of operations which are used by all the entities that want to make use of the space. Such operations are thus fixed by the interface of the space itself. The data format used by CSpaces is currently not defined and we thus provide a grounding to the operations (where possible). More precisely, we ground to *write*, *take*, *waitToTake*, *read*, *waitToRead* and *scan* operations. The current status of WSMO choreography does not provide constructs to support asynchronous calls. So operations such as *subscribe* and *advertise* cannot be modeled.

4.2.1 Grounding to CSpaces Operations

We will now describe how to ground the specified operations illustrating in the process the information required to define such a grounding. As a reference, we will use the API defined by CSpaces (please refer to table 4)

The write operation is defined as follows:

```
void  write (set tuples, URI cs_destination, URI cs_origin)
```

whereby the parameter "tuples" defines the set of tuples (specified using a domain theory stored on a CSpace cs_origin) that are to be written to a particular CSpace "cs_destination". Given a particular concept (or relation) *c*, the grounding information should specify how *c* will be encoded (or bound) into a tuple (or set of tuples) of the form: **<guid, fm, type, sguid, vguid, mguid>**. Furthermore, each concept that belongs to a state signature is grounded to a concrete operation and a parameter. The same concept can have a different grounding (operation + parameter) in a different state signature.

The retrieval operations are defined as follows:

```
Tuple take (Template|Query t, URI cs_destination, URI cs_origin)

Tuple waitToTake (Template|Query t, URI cs_destination,
                  URI cs_origin)

Tuple read (Template|Query t, URI cs_destination,
            URI cs_origin)

Tuple waitToRead (Template|Query t, URI cs_destination,
                  URI cs_origin)

Set   scan  (Template|Query t, URI cs_destination, URI cs_origin)
```

We will start with the *read* operation which is the simplest operation to ground. Notice though that these operations define the same parameters and thus are closely related to each other (the difference being the semantics of the individual operations). In WSMO choreography, instances of concepts and relations are implicitly read from within the condition of the transition rules. Such instances are either **monitored** (***in***), **shared, static** or **controlled**. However, only monitored (in) and shared instances are grounded. For the case of a read operation, the particular concept or relation should be mapped to the return value of the operation (which in this case is a tuple). The template *t* is in fact the condition of the transition rule since, as defined in [Scicluna et al., 2006], the condition of a transition rule can be regarded as queries over the state ontologies. The URIs "cs_destination" and "cs_origin" are only known at runtime and thus it is up to the particular agent implementation to define such parameters. The rest of the operations are grounded in the same way, however, note that current version of WSMO Choreography cannot distinguish between asynchronous and synchronous communication.

4.2.2 Grounding Ontology

In order to be able to express the necessary grounding information for CSpaces, a grounding ontology has been defined. Such ontology describes the tuplespace operations defined above and also the concepts necessary to encode the grounding information. We assume that there exists an ontology which describes the constructs defined in the CSpace (such as tuples, operations, etc.). Optionally, the grounding ontology presented here may

be integrated directly with such an ontology. For the sake of conciseness and clarity, we will hereby provide snippets from the ontology and refer to Appendix I for the complete version.

The ontology first defines the upper *operation* and *parameter* concepts. Each of these concepts defines an attribute *name*. In both cases, these are used by the grounding information (defined also as a concept) in order to allow the designer of the semantic web service to specify the operation and the parameter to which a particular concept or relation binds to. An axiom *operationNames* defines the names of the operations that can be used within instances of the concept *operation*. Each sub-concept of parameter must implement its own axiom to restrict the use of this attribute.

```
concept csOperation
  nonFunctionalProperties
    dc#relation hasValue operationNames
  endNonFunctionalProperties
  name ofType (1) _string

axiom operationNames
  nonFunctionalProperties
    dc#description hasValue "Defines the names of operations"
  endNonFunctionalProperties
  definedBy
    forall {?operation}
      (?operation[
        name hasValue ?operName
      ] memberOf csOperation implies
      ?operName = "write" or
      ?operName = "take" or
      ?operName = "waitToTake" or
      ?operName = "read" or
      ?operName = "waitToRead" or
      ?operName = "scan" or
      ?operName = "countN" or
      ?operName = "subscribe" or
      ?operName = "unsubscribe" or
      ?operName = "advertise" or
      ?operName = "unadvertise" or
      ?operName = "getTransaction" or
      ?operName = "beginTransaction" or
      ?operName = "commitTransation" or
      ?opername = "rollbackTransaction"
      ).

concept parameter
    nonFunctionalProperties
    dc#description hasValue "Defines the common elements
                            of a parameter for an operation"
    endNonFunctionalProperties
    name ofType (1) _string
```

Listing 4.4: operationName and parameter concepts

An example of an operation is shown in listing 4.5 below. This particular *subscribe* operation defines an agent (*paramAgent*), a template or a query (*paramTemplateOrQuery*), a callback (*paramCallback*) and CSpace destination and origin uri (*paramCs*) as parameters. Note that for the second parameter, it is not yet been clarified whether this operation will use a template or query object. For this purpose, the ontology defines an upper concept *paramTemplateOrQuery* which is the super-concept of *paramTemplate* and *paramQuery* such that the type is inferred at reasoning time.

```
concept subscribeOperation subConceptOf csOperation
    agent ofType (1) paramAgent
    templateOrQuery impliesType(1) paramTemplateOrQuery
    callback ofType (1) paramCallback
    csUriDestination ofType (1) paramCs
    csUriOrigin ofType (1) paramCs
```

Listing 4.5: An example of an operation for triple space

Finally, the *groundingInformation* concept is defined. A particular Semantic Web Service designer would create an instance of this concept which defines the necessary information for all the concepts and relations that are to be grounded.

```
concept groundingInformation
  operation ofType (1) csOperation
  bindingParameter ofType (1) parameter
```

Listing 4.6: groundingInformation concept

4.2.3 VTA Example

We will now consider a simple example of a choreography transition rule of a Virtual Travel Agency choreography description. To keep the document concise, we will limit ourselves to describe the choreography and the respective transition rule as shown in Listing 4.7. The simple choreography accepts a *reservationRequest* which defines the start and end locations, and departure and return dates for a trip. If such a trip exists, a *reservationOffer* is returned to the client.

```
choreography VTAChoreography
  stateSignature vtaSignature
    importsOntology {_"http://www.example.org/vta/vtaOntology‟}

    in
      tr#reservationRequest withGrounding

_"http://www.example.org/vta/vtaCsGrounding#reservationRequestGrounding"

    out
      tr#reservationOffer withGrounding
_"http://www.example.org/vta/vtaCsGrounding#reservationOfferGrounding"
```

```
  transitionRules vtaTransitionRules
    forall {?request} with
      (?request[
        startLocation hasValue ?startLocation,
        endLocation hasValue ?endLocation,
        departureDate hasValue ?departureDate,
        returnDate hasValue ?returnDate
      ] memberOf tr#reservationRequest and
      exists {?trip} (?trip[
        source hasValue ?startLocation,
        destination hasValue ?endLocation,
        departure hasValue ?departureDate,
        return hasValue ?returnDate) do
      add(_#[
        trip hasValue ?trip
        ] memberOf tr#reservationOffer)
    endForall
```

Listing 4.7: VTA Choreography Example grounded to CSpace

The state signature of the choreography imports state ontology of the VTA (which we will assume it exists). Furthermore, the *in* and *out* concepts are defined which are linked to an ontology defining the respective grounding information. The *reservationRequest* is grounded to *read* operation and bounded to the *tuple* parameter which is in fact the return value of the operation. The *reservationOffer* is grounded to the *write* operation and bounded to the *tuple* parameter. Listing 4.8 shows the grounding ontology for VTA.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-core"

namespace { _"http://www.example.org/vta/vtaCsGrounding#",
  dc _"http://purl.org/dc/elements/1.1#",
  cs _"http://www.example.org/cs/CSpace#",
  tsg _"http://www.example.org/cs/csGroundingOntology#"
}

ontology _"http://www.example.org/CSpace/vtaCsGrounding"
  nonFunctionalProperties
    dc#creator hasValue {"James Scicluna"}
    dc#contributor hasValue {"Francisco J. Martin-Recuerda", "James
Scicluna"}
    dc#description hasValue {"An example of grounding the
                              VTA Choreography to CSpace"}
  endNonFunctionalProperties

  importsOntology{
    _"http://www.example.org/tsc/csGroundingOntology",
    _"http://www.example.org/tsc/CSpace"
  }

  /*
   * Grounding reservationRequest
```

```
    */
  instance reservationReadOperation memberOf csg#operation
    name hasValue "read"

  instance reservationParameter memberOf csg#parameter
    name hasValue "t"

  instance reservationRequestGrounding memberOf
csg#groundingInformation
    operationName hasValue reservationReadOperation
    bindingParameter hasValue reservationParameter

  /*
   * Grounding reservationOffer
   */
  instance offerWriteOperation memberOf csg#operation
    name hasValue "write"

  instance offerParameter memberOf csg#parameter
    name hasValue "tuple"

  instance reservationOfferGrounding memberOf
csg#groundingInformation
    operationName hasValue offerWriteOperation
    bindingParameter hasValue offerParameter
```

Listing 4.8: Grounding Ontology of VTA

5 Related work

Semantic tuplespace computing initiatives (mainly sTuples, Triple Space Computing, Semantic Web Spaces and CSpaces) aim to promote a new generation of middleware infrastructures that exploit the benefits of machine processable semantics and complement current semantic web services initiatives. In this section we will provide an overview of relevant middleware technologies.

Middleware is the "*glue*" that facilitates and manages the interaction between applications across heterogeneous computing platforms. A common approach to achieve this goal is usually offering programming abstractions that hide some of the complexities of building distributed application [Alonso et. al., 2004].

Remote Procedure Call (RPC) [Birrell and Nelson, 1984] is the most basic form of middleware. It is based on synchronous method invocation and provides the necessary infrastructure to transform procedure calls in a uniform and transparent manner [Alonso et. al., 2004]. To ensure reliability in the context of multiple remote procedure calls, several extensions for RPC infrastructure were proposed for transaction support.

A *transaction* [Gray and Reuter, 1993] is a set of operations with the properties ACID (atomicity, consistency, isolation and durability). One of the most successful architectures, and the dominant form of middleware in the previous decades, was *Transaction Processing* (TP) *Monitor* [Gray and Reuter, 1993]. Built on top of Database Management Systems (*TP-lite*) or as specialized Operating Systems (*TP-heavy*), TP Monitors guarantee the successful execution of each RPC, or if there is an error, the rolled back of these operations where the systems affected are brought them to a previous consistent state (undone). RPC and TP monitor technologies were adapted to support object-oriented programming paradigm. As a result of this evolution, *Object Brokers* and *Object Monitor* were created to extend RPC and TP monitor infrastructures, respectively.

On the other hand, the necessity to support asynchronous interaction drives the evolution of the Middleware infrastructure from RPC into *Message-Oriented Middleware* (MOM) infrastructure.

MOM enables message-based interoperability where clients and service providers communicate by exchanging messages. Besides a complete asynchronous communication, MOM also balances message flows between participants and simplifies the development of interoperable applications providing support for managing errors and system failures. Among these, one of the most important abstractions is that of message queuing.

In a message queuing model, messages sent by MOM clients are placed into a queue, typically identified by a name, and possibly bound to a specific intended recipient. Whenever the recipient is ready to process a new message, it invokes the suitable MOM function to retrieve the first message in the queue.

Queuing messages provide many benefits. In particular, it gives recipients control of when to process messages. Recipients do not have to be continuously listening for messages and process them right away, but can instead retrieve a new message only when they can or need to process it. An important consequence is that queuing is more robust to

failures with respect to RPC or object brokers, as recipients do not need to be up and running when the message is sent.

Because MOM systems (like RPC-based systems) create point-to-point links between applications, and are thus rather static and inflexible with the regard to the selection of the queues to which messages are delivered, *Message Brokers* address the limitation providing flexibility in routing, filtering support and reducing heterogeneity through *adapters*.

Thanks to the possibility of defining application-specific routing logic, message brokers can support a variety of different message-based interaction models. The most well-known and widely adopted is the *publish/subscribe* paradigm. Instead of specifying the recipients of a message when applications send messages, they simply *publish* the messages to the middleware that handles the interaction. If an applications is interested in receiving messages of a given type must *subscribe* (register their interest) in a certain message broker. Siena [Carzaniga, 1998] and Hermes [Pietzuch, 2004] are two relevant implementations for publish-subscribe communication paradigm.

Alonso and colleagues argue the unsuitability of message brokers as a middleware for B2B [Alonso et. al., 2004]. The lack of trust between companies and the autonomy that each company wants to preserve are the main argument against a centralized middleware infrastructure like message brokers [Alonso et. al., 2004]. Web Services are described as a promising alternative that overcome the limitations of centralized middleware applications for B2B scenarios.

"*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*" [Haas and Brown, 2004]

[Haas and Brown, 2004] identified the core following functional aspects in the deployment of Web Services:

- **Discovery**: "*The act of locating a machine-processable description of a Web service related resource that may have been previously unknown and that meets certain functional criteria. It involves matching a set of functional and other criteria with a set of resource descriptions. The goal is to find an appropriate Web service-related resource.*"
- **Invocation**: "*The act of a message exchange between a client and a Web service according to the service's interface in order to perform a particular task offered by that service*."
- **Interoperation**: "*defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state (also called co-ordination or choreography).*"

- **Composition**: "*defines the implementation of the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function, i.e. the pattern of interactions that a Web service agent must follow in order to achieve its goal (also called orchestration).*"

Web Services are built over three main building blocks: service oriented architecture, redesign of middleware protocols and standardization [Alonso et. al., 2004]. *Service Oriented Architecture* (SOA) works on the assumption that the access to the functionality of the applications of a company is made by publishing the interface of them as a service that can be invoked by clients. The second block, the *redesign of middleware protocols* to work in decentralized environment in order to overcome the limitations of centralized middleware architectures in terms of trust and confidentiality. Finally, the last key block is a set of *standard languages* and protocols that eliminates the necessity of many different middleware infrastructures.

The deployment of several B2B and EAI scenarios to prove the suitability of Web Services technologies as a solution for business process integration have shown worst results that was expected. Existing technologies around Web Services like SOAP, WSDL and UDDI are themselves not sufficient to fully solve the integration problem: The integration still has to be done mostly manually and only marginal support during the construction process can be provided by tools, since these web service standards do not capture and exploit the actual semantics of Web Services.

Following the main principles that the Semantic Web introduced to extend the current Web, Semantic Web Services proposes to add machine processable semantics to Web Services in order to reduce manual efforts during the deployment and integration of distributed applications by improving automation in the location, combination and use of Web Services.

Two relevant initiatives have to be considered in this context. Chronologically, the first one is *OWL-S*, one of the most important outcomes of the DAML program, the major US-American Semantic Web research effort. The second recent alternative is *WSMO* (*Web Service Modeling Ontology*), the result of the joint effort of 50 academic and industrial partners heavily supported by the European Commission, the Science Foundation Ireland and the Austrian Government.

OWL-S[37] [OWL Services Coalition, 2003] is an upper level ontology for describing Web Services, specified using a formal ontology language called OWL. OWL-S contains the following elements: a Service Profile for service advertisements, a Service Model (process model) for describing how the services work and a Service Grounding for describing how the service can be accessed. **WSMO** [Roman et al., 2005] was proposed as a refinement and extension of the *Web Service Modeling Framework* (*WSMF*) [Fensel and Bussler, 2002]. WSMF defines a rich conceptual model for the development and the description of Web Services based in two main requirements: maximal decoupling and strong mediation. The model is built around four top level notions: Ontologies, Goals, Web Services and Mediators.

[37] http://www.daml.org/services/owl-s/1.1/overview/

The limitations that the message exchange paradigm brings to semantic web services has not only motivated the creation of semantic tuplespace computing approaches, but also other proposals for the integration of publish and subscribe functionality in Web Services. WS-Notification [Graham and Niblett, 2004] is part of the Web Service Resource Framework (WSRF) [Globus et. al., 2004], a new proposal to extend the dominant Open Grid Service Infrastructure (OGSI) ([Foster et. al., 2002], [Tuecke et. al., 2003]) by integrating Web Services technologies. The WS-Notification specification refers to a set of specifications comprising WS-BaseNotification [Graham and Niblett, 2004a], WS-BrokeredNotification [Graham and Niblett, 2004b] and WS-Topics [Graham and Niblett, 2004c]. WS-BaseNotification standardizes exchanges and interfaces for producers and consumers of notifications. WS-Brokered Notification facilitates the deployment of Message Oriented Middleware (MOM) to enable brokered notifications between producers and consumers of the notifications. WS-Topics deals with the organization of subscriptions and defines dialects associated with subscription expressions; this is used in the conjunction with exchanges that take place in WS-BaseNotification and WS-Brokered Notification. WS-Notification currently also uses two related specifications from the WSRF specification: WS-ResourceProperties [Graham, 2003] to describe data associated with resources, and WS-ResourceLifetime [Frey Graham, 2004] to manage lifetimes associated with subscriptions and publisher registrations (in WS-BrokeredNotifications).

On the other hand, WS-Eventing [Geller, 2004] can be considered as a subset of the WS-Notification specification, and more precisely, roughly equivalent to WS-BaseNotification. Differences arise between both specifications: complexity of the specifications, message definitions, delivery modes, subscription operations, Topic Space management and publishing. A detailed analysis of both proposals can be founded in [Pallickara and Fox, 2004].

To conclude this detailed overview of middleware infrastructures, we would like to mention a key component of Service Oriented Architecture (SOA) called Enterprise Service Bus (ESB [Keen et al., 2004]). ESB is a distributed infrastructure and is contrasted with solutions, such as broker technologies, which are commonly described as hub-and-spoke. ESB aims to provide in one infrastructure the three major styles of Enterprise Integration: Service-oriented, Message-driven and Event-driven architectures. However, ESB is positioned as an infrastructure component, and as such as a component that does not host or execute business logic. This is in contrast to components such as service requesters, service providers and the Business Service Choreography whose role is to handle business logic. Common ESB capabilities are listed below:

- ✓ Mediation or transformation of service messages and interactions
- ✓ Routing, Addressing, Publish / subscribe, Fire & forget, events and Synchronous and asynchronous messaging
- ✓ Authentication, Authorization, Non-repudiation, Confidentiality and end-to-end security.
- ✓ Transactions (atomic transactions, compensation, WS-Transaction

6 Conclusions and Future Work

Several achievements have been presented in this report. The first one is the identification of the limitations that the message exchange paradigm causes on current (Semantic) Web services proposals. In particular, this kind of communication requires strong coupling in terms of reference and time (synchronicity). As a side effect, [Fensel, 2004; and Bussler, 2005] already highlighted the contradiction represented by the message exchange paradigm in comparison with the core design principles of the Web.

[Fensel, 2004; Tolksdorf et al., 2004; Bussler, 2005; Martin-Recuerda, 2005; and Krummenacher et al., 2005] suggest that tuplespace computing can be the appropriate communication means to solve the limitations that the message exchange paradigm represents. Together with sTuples [Khushraj, et al., 2004], all these approaches present differences from the conceptualization and architecture point of view that has been described in detail in this report. To facilitate the analysis and latter comparison, we have identified seven main aspects for each proposal: semantic data model, organizational model, coordination model, collaborative and consensus-making model, security-trust model, knowledge access model and architecture model.

The best features of each proposal have been selected to determine a unified framework that we will use as a reference for future research and implementation efforts in Knowledge Web. Because the unified framework intends to cover the main aspects of each proposal, the result of this work can benefit in the future from each of the approaches described in this report. There are still some open issues about the definition of this unified framework that we will resolve by the new version of this report due month 36.

The final contribution of this report is a description of how choreography and orchestration can be grounded in CSpaces (and hopefully we will do the same for our unified framework for semantic enabled tuplespace computing in the next version of this document). To keep coherence with other parallel efforts in the Knowledge Web workpackage 2.4 (Semantic Web Services), we choose WSMO as a reference specification for semantic web services, and in particular, its proposal for choreography and orchestration. Our work reflects that the current WSMO specifications for choreography and orchestration are still in an early state[38], but we expect that in the next revision of this report the choreography and orchestration specifications will be in a more mature status.

Acknowledgements

The author of CSpaces would like to thank Uwe Keller (Leopold-Franzens Universität Innsbruck, Austria), Stefan Decker, Mathew Moran and Eyal Oren (National University Ireland Galway), Manfred Hauswirth (Ecole Polytechnique Federal de Lausanne, EPFL), Michael Genesereth (Stanford University), Charles Petri (Stanford University) and the members of the Stanford Logic Group for fruitful discussion and feedback. The authors

[38] The same can be applied to other related efforts in choreography and orchestration

of Semantic Web Spaces would like to credit Prof Robert Tolksdorf (FU Berlin) for his guidance in the area of Linda and tuplespaces.

Annex I

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-full"

namespace { _"http://www.example.org/cs/csGroundingOntology#",
        dc _"http://purl.org/dc/elemencs/1.1#",
        cs _"http://www.example.org/cs/cSpace#"}

ontology _"http://www.example.org/cs/csGroundingOntology"
       nonFunctionalProperties
              dc#creator hasValue {"James Scicluna"}
              dc#contributor hasValue {"James Scicluna", "Francisco J.
                                       Martin-Recuerda"}
              dc#description hasValue {"An ontology for grounding WSMO
                                       Choreography to CSpaces"}
       endNonFunctionalProperties

       importsOntology {
              _"http://example.org/cs/cSpace"
       }

       /*
        * Upper Concept Operation defines a single attribute "name"
        * defining its name. The axiom
        * "operationNames" restrics the names an operation may have.
        */
       concept csOperation
              nonFunctionalProperties
                     dc#relation hasValue operationNames
              endNonFunctionalProperties
              name ofType (1) _string

       axiom operationNames
              nonFunctionalProperties
                     dc#description hasValue "Defines the names
                                              of operations"
              endNonFunctionalProperties
              definedBy
                     forall {?operation}
                            (?operation[
                                   name hasValue ?operName
                            ] memberOf csOperation implies
                            ?operName = "write" or
                            ?operName = "take" or
                            ?operName = "waitToTake" or
                            ?operName = "read" or
                            ?operName = "waitToRead" or
                            ?operName = "scan" or
                            ?operName = "countN" or
                            ?operName = "subscribe" or
                            ?operName = "unsubscribe" or
                            ?operName = "advertise" or
                            ?operName = "unadvertise" or
                            ?operName = "getTransaction" or
```

```
                    ?operName = "beginTransaction" or
                    ?operName = "commitTransation" or
                    ?opername = "rollbackTransaction"
                    ).


/*
 * Upper parameter concept defines a single attribute "name"
 * defining the name of the paramter.
 * Each parameter of an operation is a subconcept of this concept
 * and defines its own axiom
 * which describes the name of the parameter.
 */
concept parameter
       nonFunctionalProperties
              dc#description hasValue "Defines the common elements of
                                       a parameter for an operation"
       endNonFunctionalProperties
       name ofType (1) _string

/*
 * Parameters of the Operations
 */

concept paramTuple subConceptOf parameter
       nonFunctionalProperties
              dc#relation hasValue paramTupleName
              dc#description hasValue "Defines the Tuple parameter.
                                       Note that this concept allows
                                       to define more than one tuple"
       endNonFunctionalProperties
       type ofType cs#tuple

axiom paramTupleName
       nonFunctionalProperties
              dc#description hasValue "Defines the name of the
                                       Tuple parameter"
       endNonFunctionalProperties
       definedBy
              forall {?param}
                     (?param[
                            name hasValue ?paramName
                     ] memberOf paramTuple implies
                     ?paramName = "tuples").

concept paramCs subConceptOf parameter
       nonFunctionalProperties
              dc#relation hasValue paramCsName
       endNonFunctionalProperties
       type ofType (1) _iri

axiom paramCsName
       nonFunctionalProperties
              dc#description hasValue "Defines the name of the
                                       CSpace parameter"
       endNonFunctionalProperties
       definedBy
              forall {?param}
                     (?param[
                            name hasValue ?paramName
                     ] memberOf paramCs implies
```

```
                    ?paramName = "cs_destination" or
                    ?paramName = "cs_origin").


concept paramAgent subConceptOf parameter
       nonFunctionalProperties
              dc#relation hasValue paramAgentName
       endNonFunctionalProperties
       type ofType (1) _iri

axiom paramAgentName
       nonFunctionalProperties
              dc#description hasValue "Defines the name of the
                                       agent parameter"
       endNonFunctionalProperties
       definedBy
              forall {?param}
                    (?param[
                           name hasValue ?paramName
                    ] memberOf paramAgent implies
                    ?paramName = "agent").

concept paramTemplateOrQuery subConceptOf parameter
       nonFunctionalProperties
              dc#description hasValue "A subscription and read operations
                                       can have either a template or
                                       a query. This concept is meant to
                                       be the superconcept of these two."
       endNonFunctionalProperties

concept paramTemplate subConceptOf paramTemplateOrQuery
       nonFunctionalProperties
              dc#relation hasValue paramTemplateName
       endNonFunctionalProperties
       type ofType (1) cs#template

axiom paramTemplateName
       nonFunctionalProperties
              dc#description hasValue "Defines the name of the
                                       template parameter"
       endNonFunctionalProperties
       definedBy
              forall {?param}
                    (?param[
                           name hasValue ?paramName
                    ] memberOf paramTemplate implies
                    ?paramName = "t").

concept paramQuery subConceptOf paramTemplateOrQuery
       nonFunctionalProperties
              dc#relation hasValue paramQueryName
       endNonFunctionalProperties
       type ofType (1) cs#query

axiom paramQueryName
       nonFunctionalProperties
              dc#description hasValue "Defines the name of the query
                                       parameter"
       endNonFunctionalProperties
       definedBy
              forall {?param}
```

```
                    (?param[
                           name hasValue ?paramName
                    ] memberOf paramQuery implies
                    ?paramName = "t").

concept paramCallback subConceptOf parameter
       nonFunctionalProperties
              dc#relation hasValue paramCallbackName
       endNonFunctionalProperties
       type ofType (1) cs#callback

axiom paramCallbackName
       nonFunctionalProperties
              dc#description hasValue "Defines the name of the
                                       Callback parameter"
       endNonFunctionalProperties
       definedBy
              forall {?param}
                    (?param[
                           name hasValue ?paramName
                    ] memberOf paramCallback implies
                    ?paramName = "c").


 concept paramTransaction subConceptOf parameter
       nonFunctionalProperties
              dc#relation hasValue paramTransactionName
       endNonFunctionalProperties
       type ofType (1) _iri

axiom paramTransactionName
       nonFunctionalProperties
              dc#description hasValue "Defines the name of the
                                       transaction parameter"
       endNonFunctionalProperties
       definedBy
              forall {?param}
                    (?param[
                           name hasValue ?paramName
                    ] memberOf paramTransaction implies
                    ?paramName = "txn").

concept paramSubscription subConceptOf parameter
       nonFunctionalProperties
              dc#relation hasValue paramSubscriptionName
       endNonFunctionalProperties
       type ofType (0 *) _iri

axiom paramSubscriptionName
       nonFunctionalProperties
              dc#description hasValue "Defines the name of
                                       the subscription parameter"
       endNonFunctionalProperties
       definedBy
              forall {?param}
                    (?param[
                           name hasValue ?paramName
                    ] memberOf paramSubscription implies
                    ?paramName = "sub").
```

```
/*
 * Operations of the CSpace API
 */
concept writeOperation subConceptOf csOperation
       paramTuple ofType (1) paramTuple
       cs_destination ofType (1) paramCs
       cs_origin ofType (1) paramCs

concept retrievalOperation subConceptOf csOperation
       returnTuple ofType (1) paramTuple
       template ofType (1) paramTemplate
       cs_destination ofType (1) paramCs
       cs_origin ofType (1) paramCs


concept takeOperation subConceptOf retreivalOperation

concept waitToTakeOperation subConceptOf retrievalOperation

concept readOperation subConceptOf retrievalOperation

concept waitToReadOperation subConceptOf retrievalOperation

concept scanOperation subConceptOf retreivalOperation

concept subscribeOperation subConceptOf csOperation
       returnParam ofType (1) _iri
       agent ofType (1) paramAgent
       templateOrQuery impliesType(1) paramTemplateOrQuery
       callback ofType (1) paramCallback
       cs_destination ofType (1) paramCs
       cs_origin ofType (1) paramCs

concept unSubscribeOperation subConceptOf csOperation
       returnParam ofType _iri
       paramSubscriptionUri ofType (1) paramSubscription
       templateOrQuery impliesType (1) paramTemplateOrQuery
       callback ofType (1) paramCallback
       cs_destination ofType (1) paramCs
       cs_origin ofType (1) paramCs

concept advertisementOperation subConceptOf csOperation
       agent ofType (1) paramAgent
       templateOrQuery ofType (1) paramTemplateOrQuery
       cs_destination ofType (1) paramCs
       cs_origin ofType (1) paramCs

concept advertiseOperation subConceptOf advertisementOperation

concept unAdvertiseOperation subConceptOf advertisementOperation

concept getTransactionOperation subConceptOf csOperation
       returnTransaction ofType (1) paramTransaction
       csUri ofType (1) _iri

concept transactionOperation subConceptOf csOperation
       transaction ofType (1) paramTransaction
       csUri ofType (1) paramCs

concept beginTransactionOperation subConceptOf transactionOperation
```

```
concept commitTransactionOperation subConceptOf transactionOperation

concept rollbackTransactionOperation subConceptOf transactionOperation

/*
 * A Concept used for defining the necessary grounding information.
 * Note that in both
 * attributes there is no need to use "impliesType" since the
 * semantic web service
 * designer is not required to create a whole instance of an operation
 * and the respective
 * parameters but rather to define only the name
 */
concept groundingInformation
       operation ofType (1) csOperation
       bindingParameter ofType (1) parameter
```

Bibliography

[Aberer et al., 2003] K. Aberer, P. Cudré-Mauroux, A.Datta, Z. Despotovic, M. Hauswirth, Ma. Punceva, R. Schmidt, P-Grid: A Self-organizing Structured P2P System SIGMOD Record, 32(2), September 2003.

[Aberer et al., 2005] K. Aberer, A. Datta, M. Hauswirth. A decentralized public key infrastructure for customer-to-customer e-commerce, to be published in International Journal of Business Process Integration and Management, 1(1), 2005.

[AbdulRahman, 2005] F. AbdulRahman A Framework for Decentralized Trust Reasoning (PhD thesis). 2005. http://www.cs.ucl.ac.uk/staff/F.AbdulRahman/docs/thesis-final.pdf

[Alasoud et al., 2005] A. Alasoud, V. Haarslev, N. Shiri. A Hybrid Approach for Ontology Integration. Proceedings of the 2005 VLDB Workshop on Ontologies-based techniques for DataBases and Information Systems (ODBIS-2005), Trondheim, Norway, Sept. 2, 2005.

[Alonso, et al., 2004] G. Alonso, F. Casati, H. Kuno, and V. Machiraju (2004). Web Services. Springer, 2004.

[Amgound and Kaci, 2005] L. Amgoud and S. Kaci. An argumentation framework for merging conflicting knowledge bases: The prioritized case. In 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'2005, Barcelona, 06 - 08 June 2005. LNCS, p. 527-538.

[Bawa et al., 2003] M. Bawa, B. F. Cooper, A. Crespo, N. Daswani, P. Ganesan, H. Garcia-Molina, S. Kamvar, S. Marti, M. Schlosser, Q. Sun, P. Vinograd, B. Yang. Peer-to-Peer Research at Stanford, The Peers Group. In SIGMOD Record, September 2003.

[Berners-Lee et al., 2005] T. Berners-Lee, R. Fielding, L. Masinter: Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Internet Engineering Task Force (IETF), January 2005

[Birrell and Nelson, 1984] A.D. Birrell and B.J. Nelson (1984). Implementing remote procedure calls. ACM Transactions on Computer Science, 2(1):39-59, Feb. 1984.

[Blaze et al., 1996] M. Blaze, J. Feigenbaum and J. Lacy. Decentralized Trust Management. In Proceedings 1996 IEEE Symposium on Security and Privacy, pages 164-173, May 1996

[Bonatti et al., 2005] Piero A. Bonatti, Claudiu Duma, Daniel Olmedilla, Nahid Shahmehri. An Integration of Reputation-based and Policy-based Trust Management. Semantic Web Policy Workshop in conjunction with International Semantic Web Conference, Nov. 2005, Galway, Ireland

[Bonifacio et al., 2002a] M. Bonifacio, P. Bouquet and R. Cuel. "Knowledge Nodes: the Building Blocks of a Distributed Approach to Knowledge Management". Journal of Universal Computer Science, 8(6), 652-661. 2002

[Bonifacio et al., 2002b] M. Bonifacio, P. Bouquet, G. Mameli and M. Nori. KEx: A Peer-to-Peer Solution for Distributed Knowledge Management. PAKM 2002: 490-500, 2002.

[Bonifacio et al., 2003] M. Bonifacio, P. Bouquet, G. Mameli and M. Nori Peer-Mediated Distributed Knowledge Management, 2003. http://eprints.biblio.unitn.it/archive/00000426/01/032.pdf

[Bontcheva et al., 2003] K. Bontcheva, A. Kiryakov, H. Cunningham, B. Popov, and M. Dimitrov. Semantic web enabled open source language technology. In EACL workshop on Language Technology and the Semantic Web: NLP and XML, Budapest, Hungary, 2003.

[Bontcheva, 2004] K. Bontcheva. D5.1.1 Natural Language Generation for Knowledge Access. SEKT Technical Report. 2004. http://www.sekt-project.com/

[Bontcheva and Wilks, 2004] K. Bontcheva and Y. Wilks. Automatic Report Generation from Ontologies: the MIAKT approach. In Nineth International Conference on Applications of Natural Language to Information Systems (NLDB'2004), 2004.

[Bontcheva, 2005] K. Bontcheva. Generating Tailored Textual Summaries from Ontologies. Second European Semantic Web Conference (ESWC 2005). Crete. 2005

[Borgida and Serafini, 2003] A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. Journal of Data Semantics, 1:153–184, 2003.

[Borst, 1997] W. N. Borst. Construction of Engineering Ontologies for Knowledge Sharing and Reuse. PhD thesis, University of Twente, Enschede, NL, 1997

[Bryce and Cremonini, 2001] C. Bryce and M. Cremonini. Coordination and Security on the Internet. Coordination of Internet Agents: Models, Technologies, and Applications 2001:274-298

[Bull et al., 1992] J. A. Bull, L. Gong, and K. R. Sollins. Towards security in an open systems federation," in European Symposium on Research in Computer Security (ESORICS), pp. 3-20, 1992.

[Bussler, 2005] C. Bussler, DERI-TR-2005-04-22 A Minimal Triple Space Computing Architecture, April 2005. http://www.deri.at/publications/techpapers/

[Cabrera et al., 2004a] L. Cabrera, G. Copeland, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, D. Langworthy, A. Nadalin, D. Orchard, I. Robinson, J. Shewchuk, T. Storey: Web Services Coordination (WS-Coordination), November 2004. ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf

[Cabrera et al., 2004b] L. Cabrera, G. Copeland, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, D. Langworthy, A. Nadalin, D. Orchard, I. Robinson, J. Shewchuk, T. Storey, S. Thatte: Web Services Atomic Transaction (WS-AtomicTransaction), November 2004 ftp://www6.software.ibm.com/software/developer/library/WSAtomicTransaction.pdf

[Cabrera et al., 2004c] L. Cabrera, G. Copeland, T. Freund, J. Johnson, J. Klein, D. Langworthy, F. Leymann, D. Orchard, I. Robinson, T. Storey, S. Thatte: Web Services Business Activity Framework (WS-BusinessActivity), November 2004. ftp://www6.software.ibm.com/software/developer/library/WS-BusinessActivity.pdf

[Calvanese et al., 2002a] D. Calvanese, G. De Giacomo, and M. Lenzerini. Description logics for information integration. In A. Kakas and F. Sadri, editors, Computational Logic: Logic Programming and Beyond, volume 2408 of Lecture Notes in Computer Science, pages 41–60. Springer, 2002.

[Calvanese et al., 2002b] D. Calvanese, G. De Giacomo, and M. Lenzerini. A framework for ontology integration. In Isabel Cruz, Stefan Decker, Jerome Euzenat, and Deborah McGuinness, editors, The Emerging Semantic Web, pages 201–214. IOS Press, 2002.

[Carzaniga, 1998] A. Carzaniga "Architectures for an Event Notification Service Scalable to Wide-area Networks". PhD Thesis. Politecnico di Milano. December, 1998.

[Christensen et al., 2001] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, March 2001.

[Cook and Brown, 1999] S. Cook and J. Brown, J. Bridging Epistemologies: The Generative Dance Between Organizational Knowledge and Organization Knowing, Organization Science, Vol.10, n°4, 1999, pp. 381-400, 1999.

[Damiani, di Vimercati et al., 2002] E. Damiani, S.D.C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. 9th ACM Conference on Computer and Communications Security, Washington DC. 2002

[de Bruijn and Polleres, 2004] J. de Bruijn and A. Polleres. Towards and ontology mapping language for the semantic web. Technical Report DERI-2004-06-30, DERI, June 2004

[de Bruijn et. al, 2004] de Bruijn, J., Martín-Recuerda, F., Manov D. and Ehrig, M.: State-of-the-art survey on Ontology Merging and Aligning V1. Project Deliverable d4.2.1, 2004. SEKT project IST-2003-506826 (http://sekt.semanticweb.org/)

[Eppler and Burkard, 2004] M. J. Eppler and R. A. Burkard. Knowledge Visualization. Towards a New Discipline and its Fields of Application, ICA Working Paper #2/2004, University of Lugano, Lugano.2004

[Eugster et al., 2001] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. Technical Report. 2001.

[Euzenat, 1995] J. Euzenat, Building consensual knowledge bases: context and architecture, in N. Mars (ed.), Towards very large knowledge bases, IOS press, Amsterdam (NL), pp143-155, 1995

[Fensel and Bussler, 2002] D. Fensel and C. Bussler. The web service modeling framework WSMF. Electronic Commerce Research and Applications, 1(2):113-137, 2002.

[Fensel, 2004] D. Fensel: Triple-based Computing. Digital Enterprise Research Institute (DERI) Technical Report DERI-TR-2004-05-31. 2004

[Fensel, 2005] Dieter Fensel. "Semantically Empowered Service-Oriented Architectures". DERI-TR-2005-07-27. July 2005. http://www.deri.at/publications/techpapers/documents/DERI-TR-2005-07-27_01.pdf

[Feier and Domingue, 2005] C. Feier and J. Domingue. D3.1v0.2 WSMO Primer WSMO Working Draft 01 April 2005. http://www.wsmo.org/TR/d3/d3.1/v0.2/

[Fielding, 2000] R. T. Fielding. Architectural styles and the design of network-based software architectures. PhD Thesis, University of California, Irvine, 2000.

[Frey Graham, 2004] J. Frey and S. Graham (editors). Web Services Resource Lifetime (WS-ResourceLifetime) Version 1.1. Technical Specification. 2004. http://www.ibm.com/developerworks/library/ws-resource/ws-resourcelifetime.pdf

[Garshold, 2003] L. M. Garshol. Living with topic maps and RDF. In the proceedings of XML Europe 2003, 5-8 May 2003, organized by IDEAlliance, London, UK.

[Gelernter, 1985] D. Gelernter. Generative Communication in Linda. ACM Transactions on Programming Languages and Systems, 7(1):80–112, 1985.

[Geller, 2004] A. Geller (editor). Web Services Eventing (WS-Eventing). Technical Specification. August 2004. http://www-106.ibm.com/developerworks/webservices/library/specification/ws-eventing/

[Globus et. al., 2004] Globus Alliance, IBM and HP. The Web Services Resource Framework (WSRF). http://www.globus.org/wsrf/

[Gong, 1989] L. Gong, "A secure identity-based capability system," in Proceedings of the IEEE Symposium on Security and Privacy, (Los Angeles, CA), pp. 55-63, IEEE, IEEE Computer Society Press, May 1989.

[Graham, 2003] S. Graham (editor). Web Services Resource Properties (WS-ResourceProperties) Version 1.1. Technical Specification. 2003. http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourceproperties.pdf

[Graham and Niblett, 2004] S. Graham and P.Niblett (editors). Web Services Notification (WS-Notification). Technical Specification. 2004. http://www-106.ibm.com/developerworks/library/specification/ws-notification/

[Graham and Niblett, 2004a] S. Graham and P.Niblett (editors). Web Services Base Notification (WS-BaseNotification). Technical Specification. 2004. http://www-106.ibm.com/developerworks/library/specification/ws-notification/

[Graham and Niblett, 2004b] S. Graham and P.Niblett (editors). Web Services Brokered Notification (WS-BrokeredNotification). Technical Specification. 2004. http://www-106.ibm.com/developerworks/library/specification/ws-notification/

[Graham and Niblett, 2004c] S. Graham and P.Niblett (editors). Web Services Topics (WS-Topics). Technical Specification. 2004. http://www-106.ibm.com/developerworks/library/specification/ws-notification/

[Grandison and Sloman, 2000] T. Grandison and M. Sloman, M. (2000). "A Survey of Trust in Internet Applications." IEEE Communications Surveys 3(4).

[Grau et al., 2004] B. Cuenca Grau, B. Parsia, and E. Sirin. Working with multiple ontologies on the semantic web. In Proceedings of the Third Internatonal SemanticWeb Conference (ISWC2004), volume 3298 of Lecture Notes in Computer Science, 2004.

[Gray and Reuter, 1993] J. Gray and A. Reuter (1993). Transaction processing and concepts and techniques. Morgan Kauffman. 1993

[Grosof, et al., 2003] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In Proc. of the Twelfth International World Wide Web Conference (WWW 2003), pages 48-57. ACM, 2003.

[Gruber, 1993] T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, Formal Ontology in Conceptual Analysis and Knowledge Representation, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[Gupta, Judge et al., 2003] M. Gupta, P. Judge and M. Ammar. A Reputation System for Peerto-Peer Networks. Thirteenth ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), Monterey, California. 2003

[Gurevich, 1995] Y. Gurevich. Evolving algebras 1993: Lipari guide, pages 9-36. Oxford University Press, Inc., 1995

[Gutiérrez et al., 2004] C. Gutiérrez; E. Fernández-Medina; M. Piattini. April 2004. International Workshop on Security in Information Systems WOSIS, April 2004, Porto, Portugal

[Haas and Brown, 2004] H. Haas and A. Brown (2004). Web Services Glossary. 2004. http://www.w3.org/TR/ws-gloss/

[Halevy, 1999] A.Y. Halevy, Combining artificial intelligence and databases for data integration, in: M. Wooldridge, M.M. Veloso (Eds.), Artificial Intelligence Today: Recent Trends and Developments, Lecture Notes in Comput. Sci., Vol. 1600, Springer, Berlin, 1999, pp. 249--268

[Hartmann et al., 2005] J. Hartmann, Y. Sure, P. Hasse, M. Suárez-Figueroa, R. Studer, A. Gómez-Pérez, R. Palma Ontology Metadata Vocabulary and Applications. In Robert Meersman, International Conference on Ontologies, Databases and Applications of Semantics. In Workshop on Web Semantics (SWWS). October 2005.

[Hayes, 2004] P. Hayes (editor). RDF Semantics. W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-mt/

[Hayton, 1996] R. Hayton. OASIS: An Open Architecture for Secure Interworking Services. PhD thesis, Univeristy of Cambridge Computer Laboratory, June 1996. Technical Report No. 399.

[Hull and Zhou, 1996] R. Hull, G. Zhou. A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches, In Proc. ACM SIGMOD '96, Montreal, Canada, 1996.

[IBM and Microsoft, 2002] IBM and Microsoft. Security in a Web Services World: A Proposed Architecture and Roadmap - technical whitepaper 7 April 2002. See http://msdn.microsoft.com/ws-security/

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

[Josang and Ismail, 2002] A. Josang and R. Ismail. The Beta Reputation System. 15th Bled Electronic Commerce Conference, Bled, Slovenia. 2002

[Kagal et al., 2001] L. Kagal, S. Cost, T. Finin, and Y. Peng. A framework for distributed trust management. Second Workshop on Norms and Institutions in MAS, Autonomous Agents, Montreal, Canada, May 29th, 2001.

[Kamvar, Schlosser et al., 2003] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In Proceedings of the Twelfth International World Wide Web Conference, May, 2003.

[Keen et al., 2004] M. Keen, S. Bishop, A. Hopkins, S. Milinski, C. Nott, R. Robinson, J. Adams, P. Verschueren, A. Acharya. Patterns: Implementing an SOA using an Enterprise Service Bus. IBM Redbooks. ISBN-0738490008. 25 July 2004. http://www.redbooks.ibm.com/redbooks/pdfs/sg246346.pdf

[Khare and Taylor, 2004] R. Khare and R. N. Taylor. "Extending the Representational State Transfer (REST) Architectural Style for Decentralized Systems." Proceedings of the International Conference on Software Engineering (ICSE), May, 2004, Edinburgh, Scotland.

[Khushraj, et al., 2004] D. Khushraj, O. Lassila and T. Finin. sTuples: Semantic Tuple Spaces", in Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous'04)

[Kopecky et al., 2005] J. Kopecky, M. Moran, D. Roman, A. Mocan. D24.2v0.1. WSMO Grounding. WSMO Working Draft 16 September 2005.
http://www.wsmo.org/TR/d24/d24.2/v0.1/

[Kotis and Vouros, 2003] K. Kotis and G. Vouros. Human Centered Ontology Management with HCONE. IJCAI'03, Ontologies and Distributed Systems Workshop, Acapulco, Mexico. CEUR-WS.org/Vol. 71, ISSN 1613-0073, 2003

[Klyne and Carroll, 2004] G. Klyne and J. Carroll, Editors. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/rdf-concepts/

[Kohl and Neuman, 1993] J. Kohl and C. Neuman, "RFC 1510: The Kerberos Network Authentication Service (V5)" RFC 1510, the Internet Engineering Task Force, Sept. 1993.

[Krummenacher et al., 2005] R. Krummenacher, M. Hepp, A. Polleres, C. Bussler, D. Fensel. WWW Or What Is Wrong with Web Services. In Proceedings of the European Conference on Web Services (ECOWS 2005), November 2005

[Lamping et al., 1995] J. Lamping, R. Ramana Rao and P. Pirolli (1995). A Focus+context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. Conference on Human Factors in Computing Systems archive. In Proceedings of the SIGCHI conference on Human factors in computing systems. Denver, Colorado, United States. 401 - 408. 1995. ISBN: 0-201-84705-1

[Lampson et al., 1992] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: Theory and practice," ACM Transactions on Computer Systems, vol. 10, pp. 265-310, Nov. 1992.

[Lee et al., 2003] S. Lee, R. Sherwood, B. Bhattacharjee. Cooperative peer groups in NICE. IEEE Infocom, San Francisco, USA. 2003

[Li et al., 2002] N. Li, J. C. Mitchell and W. H. Winsborough. Design of a role-based trust management framework. IEEE Symposium on Security and Privacy, Oakland, California. 2002

[Li and Jiang, 2004] H. Li and G. Jiang. Semantic message oriented middleware for publish/subscribe networks. Proceedings of the SPIE, Volume 5403, pp. 124-133 (2004).

[MacGregor and Ko, 2003] R. MacGregor and I.Y. Ko. Representing Contextualized Data using Semantic Web Tools. In Practical and Scalable Semantic Systems (workshop at 2nd ISWC), 2003.

[Martin-Recuerda, 2005] F. Martín-Recuerda. Towards CSpaces: A new perspective for the Semantic Web. In Proceedings of the 1st International IFIP/WG12.5 Working Conference on Industrial Applications of Semantic Web (IASW 2005). Jyvaskyla, Finland. August, 2005

[Martin-Recuerda and B. Sapkota, 2005] F. Martin-Recuerda and B. Sapkota (editors), D21.v0.1 WSMX Triple-Space Computing, 2005, http://www.wsmo.org/TR/d25/d25.1/v0.1/

[McBrien and Poulovassilis, 2003] P. J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In Proceedings of ICDE (IEEE) 2003.

[Merrick and Wood, 2000] I. Merrick and A. Wood. Coordination with Scopes, SAC (1) 2000: 210-217

[Needham and Schroeder, 1978] R. M. Needham and M. D. Schroeder, Using encryption for authentication in large networks of computers," Communications of the ACM, vol. 21, no. 12, pp. 993-999, 1978.

[O'Sullivan et al., 2002] J. O'Sullivan, D. Edmond, and A. ter Hofstede. What is a service?: Towards accurate description of non-functional properties. Distributed and Parallel-Databases, 12(2-3):117-133, 2002.

[OWL Services Coalition, 2003] OWL Services Coalition (2003). OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.0/owl-s.pdf

[Pallickara and Fox, 2004] S. Pallickara and G. Fox (2004). An Analysis of Notification Related Specifications for Web/Grid applications. Technical Report. October 30 2004

[Park and Cheyer, 2005] J. Park and A. Cheyer. Just For Me: Topic Maps and Ontologies. International Workshop on Topic Map Research and Applications (TMRA'05). Leipzig, Germany. Oct 2005.

[Paslaru-Bontas et al., 2005a] E. Paslaru Bontas, L. J. B. Nixon and R. Tolksdorf: A Conceptual Model for Semantic Web Spaces. FU Berlin Technical Report B-05-14, 2005.

[Paslaru-Bontas et al., 2005b] E. Paslaru Bontas, L. J. B. Nixon and R. Tolksdorf: Using Semantic Web Spaces to Realize Ontology Repositories. FU Berlin Technical Report B-05-15, 2005.

[Paslaru-Bontas, 2005] E. Paslaru Bontas. Using Context Information to Improve Ontology Reuse Doctoral Workshop at the 17th Conference on Advanced Information Systems Engineering CAiSE'05.

[Pietzuch, 2004] P. R. Pietzuch. "Hermes: A Scalable Event-Based Middleware". Ph.D. Thesis, Computer Laboratory, Queens' College, University of Cambridge, February 2004.

[Pujol et al., 2002] J. M. Pujol, R. Sanguesa, and J. Delgado. Extracting reputation in multi agent systems by means of social network topology. First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, pages 467--474, 2002.

[Reiter and Dale, 2000] E. Reiter and R. Dale. Building Natural Language Generation Systems. Cambridge University Press, Cambridge, England, 2000.

[Rescorla and Schiffman , 1999] E. Rescorla, A. Schiffman, The Secure HyperText Transfer Protocol. RFC 2660, Internet Engineering Task Force (IETF), August 1999

[Rhea et al., 2003] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the OceanStore Prototype. Appears in Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03), March 2003

[Roman et al., 2005] D. Roman, H. Lausen, and U. Keller (eds.): Web Services Modeling Ontology Standard. WSMO Working Draft v1.2, 2005. http://www.wsmo.org/TR/d2/v1.2/

[Sabater and Sierra, 2002] J. Sabater and C. Sierra. Reputation and social network analysis in multi-agent systems. First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy. 2002

[Scicluna et al., 2006] J. Scicluna, A. Polleres, D. Roman (editors) D14v0.2: Ontology-based Choreography and Orchestration of WSMO Services WSMO Working Draft 13 January 2006. http://www.wsmo.org/TR/d14/v0.2/

[Serafini et al., 2005] L. Serafini, H. Stuckenschmidt and H. Wache A Formal Investigation of Mapping Languages for Terminological Knowledge Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-05

[Stollberg, 2005] M. Stollberg (editor). D17v0.2: WSMO Tutorial. WSMO Working Draft 15 December 2005. http://www.wsmo.org/TR/d17/v0.2/

[Stollberg and Lara, 2004] M. Stollberg and R. Lara (editors). D3.3 v0.1 WSMO Use Case "Virtual Travel Agency". WSMO Working Draft 19 November 2004. http://www.wsmo.org/2004/d3/d3.3/v0.1/

[Suryanarayana and Taylor, 2004] G. Suryanarayana and R. Taylor. A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications. ISR Technical Report UCI-ISR-04-6, July 2004. http://www.isr.uci.edu/tech_reports/UCI-ISR-04-6.pdf

[Tolksdorf and Menezes, 2003] R. Tolksdorf and R. Menezes. Using Swarm Intelligence in Linda systems", in Proceedings of the 4th International Workshop on Engineering Societies in the Agents World ESAW03, 2003.[Tolksdorf et al, 2004] R. Tolksdorf, L. J. B. Nixon, F. Liebsch, D. Minh Nguyen and E. Paslaru Bontas. Semantic Web Spaces. FU Berlin Technical Report B-04-11, 2004.

[Tolksdorf et al, 2005a] R. Tolksdorf, L. J. B. Nixon and E. Paslaru Bontas, D. Minh Nguyen and F. Liebsch. Enabling real world Semantic Web applications through a coordination middleware. 2nd European Semantic Web Conference ESWC2005, Heraklion, Crete, May 2005

[Tolksdorf et al, 2005b] R. Tolksdorf, E. Paslaru-Bontas and L. J. B. Nixon. Towards a tuplespace-based middleware for the Semantic Web. IEEE/WIC/ACM International Conference on Web Intelligence WI2005, Compiegne University of Technology, France, September 2005

[Tolksdorf et al, 2006] R. Tolksdorf, E. Paslaru-Bontas, L. J. B. Nixon. A Co-ordination Model for the Semantic Web. ACM Symposium on Applied Computing, Coordination Models, Languages and Application Track SAC2006 (to be published).

[Tuecke et. al., 2003] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling (2003). Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum Draft Recommendation, 6/27/2003

[Ullman, 1997] J. D. Ullman: Information Integration Using Logical Views. ICDT 1997: 19-40

[Uschold, 2000] M. Uschold. Creating, integrating, and maintaining local and global ontologies. In Proceedings of the First Workshop on Ontology Learning (OL-2000) in

conjunction with the 14th European Conference on Artificial Intelligence (ECAI-2000), Berlin, Germany, August 2000.

[Visser and Cui, 1998] P. R. S. Visser and Z. Cui. On accepting heterogeneous ontologies in distributed architectures. In Proceedings of the ECAI98 workshop on applications of ontologies and problem-solving methods, Brighton, UK, 1998.

[Wikipedia, 2004] Wikipedia (2004). Semantic Web Definition. http://en.wikipedia.org/wiki/Semantic_Web

[Wilcock, 2003] G.Wilcock. Talking OWLs: Towards an Ontology Verbalizer. In *Human LanguageTechnology for the SemanticWeb andWeb Services, ISWC'03*, pages 109–112, Sanibel Island, Florida, 2003.

[Wilcock and Jokinen, 2003] G. Wilcock and K. Jokinen. Generating Responses and Explanations from RDF/XML and DAML+OIL. In Knowledge and Reasoning in Practical Dialogue Systems, IJCAI-2003, pages 58–63, Acapulco, 2003.

[Wyckoff, 1998] P. Wyckoff. TSpaces. IBM Systems Journal, volume 37, number 3, August 1998. http://www.research.ibm.com/journal/sj/373/wyckoff.html

[Xu and Embley, 2004] L. Xu, D. W. Embley: Combining the Best of Global-as-View and Local-as-View for Data Integration. ISTA 2004: 123-136

[Yao, 2003] W. Yao. Fidelis: A Policy-Driven Trust Management Framework. First International Conference on Trust Management, Crete, Greece, pages 301-307. 2003

[Yu et al., 2001] T. Yu, M. Winslett, M., and K. E. Seamons. Interoperable strategies in automated trust negotiation. 8th ACM Conference on Computer and Communications Security, Philadelphia, USA. 2001

[Zacharia and Maes, 2000] G. Zacharia, and P. Maes. "Trust Management Through Reputation Mechanisms." Applied Artificial Intelligence 14: 881-907. 2000

[Zacharia and Maes, 1999] G. Zacharia and P. Maes. Collaborative Reputation Mechanisms in Electronic Marketplaces. 32nd Hawaii International Conference on System Sciences, Hawaii. 1999

[Zaremba and Moran, 2005] M. Zaremba and M. Moran. D13.4v0.3 WSMX Architecture. WSMX Working Draft 12-10-2005. http://www.wsmo.org/TR/d13/d13.4/