TripCom

Triple Space Communication

FP6 – 027324

Deliverable

D2.1

Representing RDF semantics in tuples

Elena Paslaru Bontas Simperl (FUB)
Lyndon J. B. Nixon (FUB)
Reto Krummenacher (LFUI)
Vassil Momtchev (ONTO)
Henar Muñoz (TID)

March 27, 2007

Executive Summary

This deliverable describes the design, implementation and evaluation of an RDF-enabled tuplespace. We introduce the most representative approaches to semantics-aware coordination systems and analyze their suitability in the context of our project. Building upon the results of this analysis we introduce a novel model for representing RDF data within tuplespaces and organize them in order to optimize communication and coordination operations. The second part of the deliverable is dedicated to the prototypical implementation of these ideas. The approach has been evaluated against the initial requirements.

Document Information

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

Deliverable	**Number**	2.1	**Title**	Representing RDF semantics in tuples
Work Package	**Number**	2	**Title**	Triple Space Knowledge Representation

Date of Delivery	**Contractual**	M12	**Actual**	31-Mar-07
Status	version 1.0		final ☒	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Elena Paslaru Bontas Simperl (FUB), Lyndon J. B. Nixon (FUB), Reto Krummenacher (LFUI), Vassil Momchev (ONTO), Henar Muñoz (TID)			
Resp. Author	Elena Paslaru Bontas Simperl		**E-mail**	simperl@inf.fu-berlin.de
	Partner	FUB	**Phone**	+49 (30) 838-75224

Abstract (for dissemination)	This deliverable describes the design, implementation and evaluation of an RDF-enabled tuplespace. We introduce the most representative approaches to semantics-aware coordination systems and analyze their suitability in the context of our project. Building upon the results of this analysis we introduce a novel model for representing RDF data within tuplespaces and organize them in order to optimize communication and coordination operations. The second part of the deliverable is dedicated to the prototypical implementation of these ideas. The approach has been evaluated against the initial requirements.
Keywords	tuplespace model, tuple model, triple, space organization

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
June, 2006	1	Elena Simperl	First draft structure
July, 2006	2	Elena Simperl	Minor modifications
September, 2006	3	Elena Simperl	Finalized first draft of the deliverable
September, 2006	4	Reto Krummenacher	Finalized first draft of tuplespace model
October, 2006	5	Elena Simperl	Added requirements, revised related work
October, 2006	6	Vassil Momchev	Added requirements
October, 2006	7	Reto Krummenachaer	Added requirements
October, 2006	8	Lyndon Nixon	Added requirements
February, 2007	9	Elena Simperl	Added first draft on introduction and conclusion
February, 2007	10	Vassil Momchev	Added first draft on implementation
February, 2007	11	Elena Simperl	Finished chapters 1 and 7
February, 2007	12	Vassil Momchev	Finished chapter 6
March, 2007	13	Vassil Momchev	Revised chapter 6 according to review
March, 2007	14	Elena Simperl	Revised chapters 1 , 4 and 7 as commented by the reviewers
March, 2007	15	Reto Krummenacher	Final chapters 2,3 and 5
March, 2007	16	Henar Munoz	Finalized Section 2.6

Project Consortium Information

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

Table of Contents

List of Abbreviations

ANSI	American National Standards Institute
BSD	Berkeley Software Distribution
DAWG	Data Access Working Group
DBMS	Database Management Systems
ER	Entity Relationship
FOAF	Friend Of a Friend
HTTP	Hyper Text Transfer Protocol
iTQL	Interactive Tucana Query Language
JRDF	Java RDF
LAN	Local Area Network
LGPL	GNU Lesser General Public Licence
N3	Notation 3
N3QL	N3 Query Language
NDM	Oracle Spatial Network Data Model
OASIS	Organization for the Advancement of Structured Information Standards
ORDI	Ontology Representation and Data Integration
OWL	Web Ontology Language
OWLIM	OWL In Memory
RDBMS	Relational DBMS
RDF	Resource Description Framework
RDFS	RDF Schema
RDQL	RDF Data Query Language
ROI	RDF Input/Output
SAIL	Storage And Inference Layer
SOFA	Simple Ontology Framework API
SOAP	Simple Object Access Protocol
SeRQL	Sesame RDF Query Language
SEQUEL	Structured English Query Language
SPARQL	SPARQL Protocol And RDF Query Language
SQL	Structured Query Language
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
WSMO	Web Service Modeling Language
XML	Extensible Markup Language
YARS	Yet Another RDF Store
YARSQL	YARS Query Language

1 Introduction

One of the major requirements for the realization of semantics-aware space technology is the design of a tuplespace model, which is able to handle data formalized using Semantic Web representation languages in an efficient and effective manner. This implies means to publish and retrieve data represented in RDF, RDFS, OWL, WSML or rules while preserving the semantics of these languages, and enabling the application of reasoning services on the triple space data. In this deliverable we take a first step towards the achievement of this goal. We describe how RDF triples can be represented and organized within tuplespaces in terms of a novel model for tuples and spaces and illustrate how these models can be realized in the context of a triple space prototype implementation which has been jointly developed in work packages 1, 2 and 3.[1]

Following the Linda paradigm a triple space system should be able to represent *semantic information* through *tuples*. The expressivity of the information representation should be aligned to the expressivity of common Semantic Web languages, while respecting their semantics, so that tuples could be mapped to and from external Semantic Web resources. Regarding Semantic Web languages, we currently focus on RDF. RDF statements can be represented in a three fielded tuple (so-called *"triples"*) of the form ⟨*subject, predicate, object*⟩. Following the RDF abstract syntax, each tuple field contains a URI (or, in the case of the object also a literal). Tuples themselves can be addressed by means of URIs, which are defined through the triple space ontology (cf. Deliverable D2.2). In this way triples sharing the same subject, predicate and object can be addressed separately, which is consistent with the Linda model. Sets of related statements (i.e. RDF graphs) are represented at the level of tuples as well. Since both RDF statements and graphs are associated to URIs in the triple space ontology, we can specify the membership relation between a graph and the statements it consists of through an additional triple which references a dedicated meta-property of the ontology: ⟨tupleURI p:partOf graphURI⟩.

A triple space is defined as a container for triples, which encapsulates the RDF statements. A triple space can be divided into virtual subspaces and physically partitioned across distributed kernels [17]. Every space is addressed using a URI, which is installed by the space owner and captured in the triple space ontology. Just as in the case of individual tuples, this URI is useful in terms of the REST communication model. A space may contain multiple (sub-)spaces, while it can only be contained in at most one parent space. The latter holds also for tuples, which are explicitly associated to a single space. In order to allow for overlapping between spaces the triple space model resorts to the notion of *"scopes"* [34]. Scopes are temporary tuple containers. Unlike subspaces, which form the virtual structure of the triple spaces, they can be created individually by clients based on arbitrary filters or constraints.

This deliverable is organized as follows. Chapter 2 provides a collection of requirements for the structure and organization of RDF data within tuplespaces identified in related work packages in the project. The requirements range from functional and architectural ones, as specified by technical work packages 1, 2, 3, 4 and 5 to more application scenario-related, as resulted from work packages 8a and 8b. Chapter 3 gives an overview of previous work in the area of Linda-based coordination systems, focusing on the design of tuplespaces and the various types of extensions from the original model arising in various application settings. Further on, the chapter con-

[1] `http://sourceforge.net/projects/tripcom/`

tains a brief analysis of existing Semantic Web frameworks, which provide interesting insights on the way heterogeneous Semantic Web data can be persistently managed within the same application in an efficient manner. Building upon the results of the aforementioned chapters, Chapters 4 and 5 perform an analysis of potential approaches to tuplespace management and organization models and introduce and motivate the TripCom tuple and tuplespace models, respectively. These concepts have been integrated into the first TripCom prototype implementation, which is described in Chapter 6. The deliverable is concluded with a summary of the tasks carried on so far and an outline of planned future work (Chapter 7).

2 Requirements analysis

This chapter provides a collection of requirements for the abstract models used to manage RDF data within tuplespaces. These requirements have emerged in several related work packages in the TripCom project, both with a technical and an application-oriented focus.

2.1 Work package 1

The storage infrastructure developed in work package 1 provides highly scalable RDF-like triple data model. The tripleset model is an extension of the RDF model [37] to allow efficient and natural association of meta-data to the statements.

The key requirements toward work package 2 are to define meta-schema to model tuples and their logical organization into spaces compatible with the designed tripleset storage data model. Another requirement is to control the distribution of the data model and manage the clusters of data stores. Also, credentials to the storage infrastructure to authenticate and authorize the current user have to be supported.

2.2 Work package 2

Work package 2 should deal with the issues of representing semantic data models such as RDF and OWL or WSML efficiently and consistency within a tuplespace. This covers modeling the tuples as well as the tuplespace semantically and devising an efficient distribution scheme for subspaces.

In accordance to these objectives the prospected *tuple model* should capture Semantic Web information in a way which is compliant with the abstract syntax and semantics specifications of the corresponding knowledge representation languages. In particular this implies that the conceptual model underlying triple spaces should preserve the semantics of RDF and should be easily extendible towards the support of other languages such as OWL, WSML or rules.

The way tuples are organized within triple spaces and the relationships between different triple spaces is known as *tuplespace model*. This model should form the basis for the application of heuristics for improving the scalability of triple spaces. It should provide the unified view upon a potentially distributed, open and dynamic Web of semantic data.

2.3 Work package 3

The access layer defined in this work package combines with the management layer defined in work package 2 to form the core of the triple space kernel, i.e. the runtime system. The access layer specifies the coordination model of triple space and a coordination language by which agents interact with the system respecting that coordination model. The management layer specifies the means by which those interactions are modelled and realized in terms of data and memory structures (tuples and tuplespaces). Hence the coordination model and language need to align with the chosen structures of the management layer (granularities of data structures, topography of

memory structures) and requires that the management layer is able to support the interactions defined by the coordination model and language.

When the triple space is distributed, interactions between the access and management layers could be more efficient if the management layer is able to provide optimization data which indicates where a certain operation (e.g. tuple emission or retrieval) could be best applied (in terms of a physical triple space kernel). Then the access layer could pass the operation onto the correct management layer (i.e. the local one or another one in the distributed system). Another aspect of triple space access is the blocking of operations and notifications. The management layer needs to provide facilities to allow the access layer to monitor tuple insertions in the space in order to determine if a retrieval or notification should take place (given that insertions do not all take place over a single kernel / access layer). Finally, the management layer should ensure the physical mobility of tuples (while retaining their virtual location) in order to optimize access (e.g. by ensuring tuples of certain types can be found close to access operations of the same type).

2.4 Work package 4

Work package 4 aims at defining triplespace communication and coordination for Semantic Web services. It is concerned with the interfaces of TripCom that will be used by Semantic Web services to communicate. There are different initiatives for Semantic Web services established. Within TripCom the focus is set on the Web Service Modeling Ontology (WSMO) as conceptual model, the Web Service Modeling Language (WSML) as formal language and the Web Service Execution Environment (WSMX) as reference implementation of Semantic Web services.

Semantic Web services as being user of TripCom need different ways of identifying the information which is to be retrieved from the triplespace, i.e. based on the content in triples and graphs which is mainly concerned with template matching, based on unique identifiers contained in metadata of triples and graphs; and based on context, i.e. get the graph published by user X. Moreover, two kinds of retrieval operations are expected to be supported by TripCom which are either blocking or non-blocking (possibly based on a given timeout). It also requires a subscription mechanism for users to get notification about change in particular triples or graphs; and advertisement mechanism to notify all the subscribed users about publishing or change of an advertised triple or graph. Methods to deal with organizing the data published in triplespace based on subspaces are also required, i.e. creating and managing subspaces. Moreover, mapping from Web Service interaction patterns supported by WSDL 1.1 and 2.0 [8, 7] to the triplespace API primitives (work package 3) is also needed to be done in order to enable the complex interactions with Web Services possible over triplespace.

2.5 Work package 5

Security and trust applies to all aspects of the triple space including the management of tuples and spaces. The means to reference individual triples as well as groups of tuples is necessary in order to be able to apply security and trust information to them. Additionally, security might be even applicable to individual resources in a tuple on one hand or an entire space of tuples on the other, hence these different levels of

granularity must be supported. The management of the security and trust information will probably be handled in the same manner as other tuples in the management layer even though the sensitivity of this information may lead to it being virtually and physically stored in some dedicated security infrastructure.

The management layer may be able to interpret internally security and trust data to govern its organization of tuples and spaces as well as individual views upon them from the access layer. For example, an agent could have a minimum trust level specified for the tuples it seeks and should only "see" tuples to which that minimum level of trust can be applied.

Finally, it is possible that data in the management layer is encrypted. Given that an agent does have right of access to that data, it must still be possible to handle the encrypted data according to its content, e.g. if the triple content matches the agent's template, that it could still be retrieved by the agent even though its content is actually encrypted.

2.6 Work packages 8a and 8b

From work package 8a [12] Digital Asset Management (DAM) use case needs to store in TS content catalogue, services, negotiation and auction messages and contracts

- Content catalogue, which involves a set of content summaries distributed that content providers trade with them.
- Services, constituted by contents, rights and distribution channels that Service Providers offer to customers.
- Negotiation and auction messages interchanged between Service Provider and suppliers in order to have a common agreement.
- Contracts cover each one of the agreements that Service Provider keeps with their supplier to provide a service.

The last three kinds of data are constituted by EDIFACT messages since the service also involves a set of EDI messages. Data domain can be represented by DAM ontology. So information will be constituted by EDIFACT ontology and DAM ontology.

From work package 8b [15] European Patient Summary (EPS) use case involves to store in the TS patient summary data which constituted by patient data and health record. Also it considers limited geographic information to support the emergency use case.

From both use cases [12][15] also authority-oriented data should be taken into account, which involves roles and security policies to determine actor permissions and to manage data.

3 Related work

In this chapter we analyze earlier work, which could be relevant for the design of the conceptual model of triple spaces. The analysis can be divided into three main directions: tuple and tuplespace models in previous Linda-based systems, conceptual design of related semantic tuplespaces and frameworks for managing semantically enabled information.

3.1 Types of tuples and tuplespaces in Linda-based systems

The purpose of this section is to provide a survey of the most important proposals to tuple and tuplespace models in the coordination literature. By contrast to the related survey work on Linda extensions, which has been done in work package 3, this survey focuses on the types of information which can be stored in a tuplespace, on the data structures used to manage this information (i.e. the tuple model) and the ways tuplespaces are organized (tuplespace model).

A conventional Linda-based coordination system foresees a virtual representation of data as tuples, which are ordered sets of typed fields [21]. The tuples are organized in tuplespaces, which are shared collections of tuples. Tuple fields are fundamentally typed using the type system of the host implementation language. The order of the fields does not carry any additional default semantics, though this can be adjusted to the needs of a specific application scenario.

These initial concepts have been revised in order to suit to particular application scenario constraints:

- Approaches proposing new types of tuplespaces aim to overcome the technical problems of dynamic, open, distributed systems (e.g. heterogeneity, scalability, fault-tolerance, coordination of multi-user access), by proposing distribution strategies and various tuplespace structures such as multiple spaces, hierarchical spaces etc.

- A second research direction extends the primitive set of field types (usually those of the host implementation language) with new types of tuples to support more complex data structures as tuple field values, introduce flexibility through rules, event models and logic-based approaches.

3.1.1 New types of tuplespaces

With respect to the structure of the tuplespaces we differentiate between *single* and *multiple space* approaches. The latter can be further classified according to the partitioning strategy and the way the spaces are interrelated:

- *Flat* multiple spaces with or without overlapping information.

- Hierarchically organized spaces (*nested* spaces).

KLAIM [13] introduces distributed tuplespaces in terms of flat collections of multiple spaces. Nested tuplespaces are addressed for instance in Bauhaus Linda [5], Melinda [26] or Polis [9].

The introduction of a multiple space paradigm is accompanied in some systems by the definition of primitives to create and manage spaces as first-class objects. This extension has consequences on the signatures and the semantics of the Linda primitives, and on the definition of the protocols by which agents residing in different spaces communicate. The way spaces are addressed depends on their structure: in case of a flat set of spaces these are assigned a *unique identifier* (as for instance in KLAIM). Nested spaces are provided with a relative identification mechanism based on the *identifiers of the parent spaces*. The communication patterns are also influenced by this distinction. A flat collection of spaces does not impose any constraints on the communication, as every space is uniquely addressed. Agents can communicate to each other within a particular space and can store and retrieve data from these spaces. In this case the Linda primitives contain an additional parameter, which identifies the actual space. Hierarchical collections allow two brother spaces to communicate by introducing and retrieving messages in the parent space, while Linda primitives act per default on local or parent spaces.

In the following we will shortly mention further tuplespace implementations which introduce particular types of tuplespaces:

- TuCSon [38] is a coordination model for Internet applications based on mobile agents. The tuplespace model introduces the notion of *"tuple centers"* containing hierarchically organized sets of spaces.
- The PageSpace approach [10, 11] deals with the realization of a generic middleware to support coordination in a multi-agent environment on the basis of the Linda paradigm extended with notions of rules and services, which are managed in different spaces. In this case the organization of the global space occurs according to content criteria, so that service information is managed separately than rules or application data.
- XMLSpaces [48, 45] uses a tree-like structure for tuplespaces. The global space organization is stored as an XML document.

3.1.2 New types of tuples

New types of tuples can be classified according to their *structure* and their *content*:

- As regarding the structure we can distinguish among *flat* and *nested* tuples. The latter allow typing tuple fields to tuples. Flat tuples can be additionally extended with an identifier field. In case of the nested ones, the identifiers are relative to the parent tuple.
- The information which is stored as tuples in the tuplespace:
  - Rules as in Prolog-D-Linda
  - Java objects as in JavaSpace
  - XML Documents as in XMLSpaces
  - Services as in Workspaces
  - Reactive tuples as in Law-Governed Linda

In the following we will give an overview of some of the most important tuplespace systems and the type of tuples:

- Prolog-D-Linda [43, 44] is an implementation of Prolog extended with Linda-style parallelism and supporting a distributed tuplespace. Tuples are expressed as Prolog clauses and tuplespaces as Prolog databases. Both Prolog facts and rules can exist in the space.
- The coordination language Laura [47] is designed for open distributed systems. It facilitates the use and offering of services which are characterized by a high degree of heterogeneity and dynamics with regard to the number of application components entering, leaving or being temporarily not available to the overall system. Here, the tuplespace containing data is replaced with a service space containing forms describing service offers, requests, and results. Matching is performed on the basis of the service interface that is included in each of these kinds of forms.
- XMLSpaces [48, 45] is an extension of Linda for Web-based applications, in which XML documents can be stored in tuple fields and matched according to XQuery patterns.
- LGL (Law-Governed Linda) [35, 36] applies the concept of law-governed architecture from the field of centralized message-passing systems to Linda in order to support the use of Linda as a coordination model for open systems.
- MARS [4] is a programmable coordination architecture for mobile agent applications, defining Linda-like tuplespaces that react with specific actions to the accesses made by the mobile agents.
- JavaSpaces [20] realizes a network repository of Java objects, which are serialized as tuples. Tuple content can be any Java object.
- TSpaces [49] follows a similar object-oriented model, with both tuple types and field names support.
- Workspaces [27] add to the classical Linda tuple self-description (giving names to fields), typing and source/destination tags (for routing in a distributed system).

3.2 Semantic tuplespace computing

After an overview of general-purpose coordination systems we now turn to a study of those ones which explicitly address the issue of storing and managing information with a machine-understandable semantics.

3.2.1 sTuples

sTuples [29] 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 persistency, 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. JavaSpaces is also integrated with the Vigil framework [28] for realizing "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 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 a 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[1], a Description Logics reasoner [23]. 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. Due to the increased complexity of different Description Logic-based queries, the matcher performs the resolution through a series of steps with increasing complexity.

3.2.2 Triple Space Computing (TSC)

Triple Space Computing [18] extends tuplespace computing, a simple and flexible coordination mechanism, using RDF as the formalism for describing the content of

[1] http://www.racer-systems.com/

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, [18] proposes the use of RDF to overcome this problem and create a natural link from the space-based computing paradigm into the Semantic Web. [30] extends the work of [18] with a concrete proposal about how to represent tuples using quads instead of triples. Accordingly, triples are uniquely identified through URIs. This means that each triple in any triplespace is uniquely marked and can be distinguished from all the other triples by its URI. In this way triples become quads [32]. Further on, [19] suggest to only use one identifier per set of triples, i.e. per RDF graph. This seems to contradict the idea of one resource one identifier common in the Web, where resources are tagged with one distinct identifier. The authors however argue that the use of having an identifier per triple does not justify the added complexity on storage and processing level. On the one hand, the identifier is used to add context information to semantic data that will not defer from one triple to another within a set that is written at the same time. On the other the URI can be used to directly address a given set of triples, which is a big advantage when exchanging whole objects, like e.g. Web service descriptions or business orders. In both cases the URI per subgraph approach is clearly sufficiently fine-grained. Moreover this approach has proven to be effective in [6].

The space structuring is defined to consist of a disjoint set of space, each identified by an own URI and providing a particular communication area; i.e. TSC proposes the inauguration of new spaces whenever there is a new topic, a new group of agents or new security aspects in consideration. This approach is envisioned to improve at least local scalability by naturally restricting the amount of participants in a given space. No space hierarchies, nor overlapping spaces are however allowed in order to simplify the prototype implementation. Any interaction with the triplespace is at all times performed against a particular space, hence there is no such concept as a global or root space defined yet. The interlinking, as well as the annotation with meta-information of spaces and graphs, is done by use of a small triplespace ontology that defines relations to express vicinity of information, the publishing entity, or the time of publication.

The space infrastructure is implemented through an extension of the CORSO (Coordinated Shared Objects) middleware [14] that already provides replication, transaction and coordination of amongst others Java objects (also .NET and C++ classes). Hence, RDF graphs are mapped onto CORSO objects in order to share them amongst participating nodes. This virtual shared memory space has already proven its effectiveness in various industrial projects. The prototype implementation of TSC makes moreover use of the YARS storage framework [24] in order to ensure persistency and RDF querying. YARS is a highly scalable RDF store that supports the use of quads by help of its support for contextualized storage. A context in YARS is a particular area in the local or remote storage. These contexts are directly used to map spaces and graphs; every space and graph is mapped to a particular context and hence also distinguishable at storage level. As, the queries to YARS are expressed in N3QL, an N3-based language, TSC defines semantic templates to be graph pattern based (cf. also [39]).

3.2.3 Semantic Web Spaces

Semantic Web Spaces [46] has been proposed by the Free University of Berlin. It was originally envisioned as an extension of their XMLSpaces work, an implementation of a tuplespace platform that extended the Linda coordination model in order to allow tuple fields to also contain XML documents and to match templates based on XPath expressions or other XML Query forms. The proposed next stage, an RDFSpaces platform supports the exchange of RDF triples in form of 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. The model of Semantic Web Spaces represents RDF information in dedicated tuples typed as RDFTuple and considers an agent to have two views upon an RDFSpace:

- 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 an 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 [34]. They improve the scalability of open distributed Linda systems, reduce the same time the system complexity by restricting operations to a specific subset of the space and enrich the interaction patterns without expanding the number of coordination primitives. Rather than using multiple or nested tuplespaces, scopes logically partition the single tuplespace into arbitrarily overlapping physical subspaces and can be considered as being a particular view upon a tuplespace in which a certain subset of the tuples of the global tuplespace are seen. An inserted tuple is associated with the scope attached to the insertion primitive; matching moreover only sees the tuples in the scopes attached to the matching primitive. Merrick and Wood [34] demonstrate furthermore how scopes can support the multiple read operation and atomic transactions.

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, Semantic Web Spaces allow them to be considered Semantic Web instances that can have information attached to them and be shared in RDF documents.

In addition to the advantages of the above mentioned scopes, contexts allow agents to operate in subspaces of the global space which contain the tuples relevant to them. Hence, agents can gather related tuples into specific contexts in order to perform specific tasks within that context. Another 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.

3.2.4 Conceptual Spaces (CSpaces)

Conceptual Spaces (CSpaces, [33]) was born as an independent initiative to extend Triple Space Computing [18] with more sophisticated features and to study their applicability in different scenarios apart from Web Services (e.g. distributed knowledge management systems [1]). A CSpace is a knowledge container defined as a set of tuples, where each tuple has a well-defined structured of seven fields

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

Ideally, fm is a first order logical formula. However, limitations imposed by applications and/or members of the CSpace can restrict fm to less expressive formalism (like description logics, or even RDF triples). The field type identifies in which formal language fm has been defined (e.g., fol, shiq dl, dlp). Unlike the Semantic Web, the current proposal of the CSpaces' semantic data model does not commit to a specific formalism until an evaluation of use cases determines which languages are most appropriate. The field subspace defines a subset of the CSpace to which a tuple belongs. Currently, there are seven different types of subspaces defined for each CSpace: domain theory, metadata, instance, trust and security, mapping and transformation rules, annotations, and subscriptions/advertisements. The field guid is a global unique id for the logical formula (which can simplify reification, and make the code more compact). The sguid field is the global unique identifier of the CSpace in which the tuple was created (which attaches provenance to a logical formula). The field vguid is a version global unique identifier for the logical formula, while the mguid field is the identifier of the member of the CSpace that stored the tuple.

Each of the seven subspaces (cf. also Table 3.1) can have a "mirror" that stores an efficient representation (in terms of reasoning performances) of the stored data. Thus, each subspace has a raw and a reasoning side. All editing operations are done in the raw side and periodically the modifications are transferred to the reasoning side. The strict separation between raw and reasoning side allows the implementation of, e.g., approximate reasoning techniques [22] like language weakening and knowledge compilation to antagonize the scalability problem and debug methods for eliminating inconsistencies [41].

The organizational model of CSpaces promotes the use of shared domain theories as an interlingua for data and application integration. A second source of inspiration for the CSpaces's organizational model is the intuition that generation and sharing of machine processable semantics will follow a bottom-up approach (from personal knowledge specifications to shared knowledge specifications). Finally, the necessity to improve trustworthiness of the information stored motivates the creation of spaces of trust for a restricted group of agents (human users and applications). Therefore two types of CSpaces were defined:

- **Individual CSpaces** are knowledge containers defined by an individual that reflects his/her own perception of a concrete domain.
- **Shared CSpaces** are conceptual spaces shared by several users that have reached an agreement on how to specify common knowledge.

Table 3.1: Details about CSpaces subspace structure

subspace	details
domain theory	stores a set of consistent logical theories which gives an explicit, partial account of a conceptualization.
metadata	provides an ontological description of the CSpace itself.
instance	used to represent individuals and the values of their attributes in a domain theory.
trust and security	described in terms of policy rules and reputation information [42].
mapping and transformation rules	defines correspondences between common terms, relations and instances of two domain theories.
annotations	defines links between concepts and instances (topics) specified in each domain theory with information resources (occurrences).
subscriptions/ advertisements	stores queries that identify the information that is requested by information consumers and will be published by information producers.

3.3 Semantic Web management tools

Orthogonal to the question of how to design a tuplespace which is able to deal with Semantic Web information, we study frameworks which have provided answers to this question in a more general context. In particular we have a closer look at Semantic Web programming environments such as Jena, Sesame and ORDI framework which offer means to create, manage and store information in RDF(S), OWL and DAML+OIL independently of the application setting in which this information is used. The conceptual models underlying these approaches share the common feature of being able to concurrently deal with heterogeneously represented Semantic Web information and are thus relevant for our goals.

3.3.1 Jena

License: BSD

Available at: `http://sourceforge.net/project/showfiles.php?group_id=40417`

Supported reasoning: RDFS; OWL-Lite; DIG 1.1 Interface

Model storage: Memory; DB; File

API Paradigm: Statement-centric; Resource-centric; Ontology-centric

Jena is Java toolkit for developing Semantic Web applications based on W3C recommendations for RDF and OWL. It provides an RDF API, RDF parser, OWL API and rule-based inference for RDFS and OWL. The toolkit supports different persistence strategies like file system or RDBMS. NG4J is an extension of Jena to support named graphs and adds the quads model operations.

3.3.2 Sesame

License: BSD

Available at: `http://www.openrdf.org`

Supported reasoning: RDFS; OWL-Lite

Model storage: Memory; DB; File; Native; RDF Repository

API Paradigm: Statement-centric; Resource-centric; Ontology-centric

Sesame is comparable framework to Jena to support persistent storage of RDF data and schema information and subsequent querying of that information, [3]. A central concept in Sesame is the "repository" interface which provides a statement and resource centric API . Repositories could be configured using the Storage And Inference Layer stacking to support security, concurrent access, versioning and RDFS or custom inference. OWLIM is very efficient and high-performance SAIL implementation to perform RDFS, OWL DLP, and OWL Horst reasoning that successfully passed the threshold of 1 billion of statements.

3.3.3 ORDI framework

License: LGPL

Available at: `http://www.ontotext.com/ordi/`

Supported reasoning: RDFS; OWL; custom rules; WSML (planned)

Model storage: Memory; Native;

API Paradigm: Statement-centric;

ORDI is ontology language neutral framework to assist the development of ontology-aware applications. Its major objectives are integration of databases and other structured data-sources, support for heterogeneous reasoners and the RDF-to-WSML interoperability through conversion of WSML to and from WSML-Triples (WSML-RDF). The ORDI framework and its successor ORDI framework second generation are fully described in the TripCom D1.2 deliverable [37].

A full overview of the available Semantic Web management tools is placed in TripCom D1.1 deliverable [40].

4 Tuple model

This chapter describes the tuple model which will be used in TripCom.

4.1 Analysis of the approaches

This section summarizes the results of the analysis about data models which were used to or at least envisioned to represent semantic information within tuple spaces in TripCom. The analysis is based on the survey about (semantically-enhanced) tuples and tuple spaces, as outlined in the previous chapter[1].

The analysis concentrates on two core topics:

1. whether the semantic data shall be represented in a serialized form across several tuple fields or within a single dedicated field in the tuple model; and

2. assuming the latter, what kind of data structure is appropriate for the representation of semantic information. In this case we address primarily strings and graphs as alternative data structures.

The difference between the string and the graph approach is first of all an implementation issue. Assuming a Java-based tuple space system, in both cases the application data is encapsulated in a single tuple field represented by a Java object. The object is naturally a Java **String** object in the former case, and a Semantic Web-specific object that models RDF graphs in the latter. Note that using strings as a universal means to encode the semantic data does consequently refrain from the need of an RDF API. In the other case the triple space implementation has to incorporate a framework to handle graphs, nodes, statements and other entities specific to the Semantic Web - one such framework would for example be provided by Jena. Although this would infer dependencies on external libraries and increase the complexity overload, only such a system could be truly considered to be a *semantic middleware* — a middleware that is explicitly aware of the semantics of the managed data, and thus able to optimize its operation with respect to the data content.

Taking these two orthogonal dimensions into account, we address hereafter the advantages and disadvantages of the following three approaches (cf. Table 5.1):

1. the semantic data is represented in terms of a single RDF graph object and established as a dedicated field within a tuple. This approach is considered in the already mentioned TSC project [19], where RDF graphs are handled using CORSO objects (Java objects with unique OIDs (object identifiers)) at space level and as contextualized quads at storage level (cf. Section 3.2.2).

2. the semantic data is represented in form of plain strings within a dedicated tuple field in a tuple, which could encode single triples, but also whole graphs or RDFS and even OWL domain knowledge.

3. the semantic data is represented in serialized form across multiple tuple fields within a tuple. In terms of RDF this implies a three-field representation of RDF statements of the form <subject, predicate, object>.

	Single field	**Multiple fields**
Complex objects	Represent semantic data as conceptual graphs	Serialize semantic data across multiple tuple fields
Plain strings	Represent semantic data as plain strings	

Table 4.1: Approaches to tuple models

In the continuation of this Section we analyze each of these three proposals and classify them according to their advantages and disadvantages.

4.1.1 Representing data as triples over three distinct tuple fields

We analyzed the implications of a tuplespace approach, in which Semantic Web data is reduced to RDF and RDF triples are serialized into three RDF resource-typed fields corresponding to the subject, predicate and object of a statement.

- **Pros**
  - Accordance with the conventional tuple concept: This RDF triple representation preserves a direct link to classical coordination models. It is compliant to the original Linda system approach, where tuples fields were typed in the type system of the implementation language. The type system of the Semantic Web, as we consider it here, is given by RDF(S) [25, 2]. In that way any data interpreter — template matcher for example — can refer directly to the RDF(S) recommendations of W3C, where the type system is explicitly formalized. There is however one restriction implied on the Linda model, as semantic data is always represented by three typed fields for RDF resources, while Linda does not make any restrictions at all on the number and type of fields.
  - Common representation language: Assuming that RDF is the core representation language of the Semantic Web, the heterogeneity at formal language level is reduced to RDF on the basis of the compatibility between this and additional Semantic Web representation languages.
  - Simple mapping to the underlying storage layer: If data needs to be persistently stored (i.e. in a triple repository), tuples can be mapped one-to-one the structures underlying these persistent storage systems.
  - Simple triple respectively resource-based template matching. Query processing simplifies to the matching of particular RDF resources or statements within given fields (e.g. ?s <#hasName> “john”).
  - Simple content indexing: The tuplespace can easily index the data according to the addressable RDF resources in one of the three fields.
- **Cons**:

[1]In this chapter we focus on tuple models, while Chapter 5 analysis the different tuple space models.

- Possibly too fine-grained: Independently of the tuplespace API (which might provide out-operations for handling single triples or sets of triples) it seems obvious that if agents tend to generally publish sets of RDF triples rather than single statements, this data model implies an additional overhead for transforming the corresponding graphs to triples and vice versa. Moreover it implies an additional overhead for the processing of queries which are not formulated according to the Linda-like field matching as shown above (e.g. queries for all classes of a given OWL ontology).
- No means to address sets of triples: In order to address sets of triples (a particular namespace, a named graph, an OWL class, rules heads and bodies etc.) the tuple model needs an additional identifier. This double contextualization (tuple identifier and container identifier) is in fact not supported by any of the existing RDF repository solutions.
- Support for additional knowledge representation languages: In order to support new knowledge representation languages (formalisms) within the tuplespace a serialization of those languages into triples (RDF) must be defined. For instance if the tuplespace needs to handle Prolog data, a triple-based representation of Prolog is required.

4.1.2 Representing data as graphs

In contrast to the first approach, the semantic data is represented with help of RDF graphs in a single field at tuple level. The difference to the former approach is that RDF statements are represented by a single graph tuple field instead of the serialization into three fields.

- **Pros**:
  - One field for the semantic data: The application data is stored within a single tuple field and is thus easier adaptable for formalisms that cannot be grounded into RDF triples.
  - No need for a second identifier: As the information is stored in form of graphs, the graph identifier is in the same time used as tuple identifier.
  - Language heterogeneity: Information is stored within the space in form of conceptualized graphs and independent of the internal language. The data is not mapped into formal language dependent tuples (such as RDF in the former approach). In consequence, the tuplespace does not have to deal with knowledge representation heterogeneity, but hands it over to respective interpretation engines.
  - Simple mapping to the underlying storage layer: If data needs to be persistently stored (i.e., in a triple repository) tuples can be easily mapped to the triple structures underlying these persistent storage systems, as the data is modeled as respective objects.
- **Cons**:
  - No direct mapping to classical coordination models. This concerns in particular also the application of template matching over the three fields of a tuple/triple.

  - Too coarse grained: In case of a discrepancy between the level of granularity of the published and consumed information, representing data as graphs could imply additional template matching overhead and graph processing.
  - Indexing: indexes are usually built at field level. If semantic data is stored within one field, the tuplespace manager needs to apply more sophisticated heuristics to build content-driven indexes. On a simple insertion operation, the tuplespace manager either processes the inserted graphs internally in order to create a useful index or indexes the information on a potentially too coarse grained basis.

4.1.3 Representing data as strings

In this third approach the tuple model foresees a single field for the application data. By contrast to the graph-oriented one, the data is represented in form of plain strings (accompanied by an additional field denoting the initial representation language).

- **Pros**:
  - No overhead for information publishing: the data is represented in a single tuple field as is.
  - One field for the semantic data: The application data is stored within a single tuple field, the tuplespace manager can thus easily distinguish between application and administrative data.
  - Language heterogeneity: Information is stored within the space without mapping it to a new formal language such as RDF. However, the respective formalism is indicated in an extra tuple field making this approach interesting for various different formalisms.
  - Inference/storage independent: The tuplespace manager is freed to adapt to different representation formalisms like RDF or graphs. The space is more flexible and independent of any type of formalism, as the processing of data is handed over to respective experts. The space uses external storage/inference systems that provide appropriate interfaces.
  - Naturally linked with the Semantic Web; Information on the Web is generally shared in form of plain text files – e.g., XML or N3 representations.
- **Cons**:
  - No direct mapping to classical coordination models: see outline in 4.1.2.
  - No semantics within the space: in order for the space to perform semantic clustering or other type of optimization, it needs to create an appropriate model of the string-represented data at run-time.
  - No means to build indexes or to define scopes without internally processing the published information, as the semantics of the information in the one field of type string is not understandable by the space. Above, the approach was highlighted as simplifying the space manager due to the hidden formalisms. This simplification however becomes a major drawback, as the interpretation of the data and thus the content-driven character of the space is lost.

Evaluation The third approach introduced does not provide any means to interpret the information within the space and epitomizes hence no option. The space manager needs at least some core knowledge about the semantics of the data. In consequence only the first (three-field tuples) and the second (a graph field tuple) are considered for further investigation.

An analysis of the remaining two alternatives with respect to the requirements derived from the other related work packages (work package 1, work package 3, work package 4, work package 8a, and work package 8b) revealed the following important points to consider:

- The clear separation of the tuplespace and the data access layers in the TripCom architecture imposes no constraints on the envisioned tuple model. The data access layer abstracts the underlying data storage framework (work package 1) entirely from the space implementation.
- Work package 3 requires a fine-grained representation of the semantic data (as in the first approach) in order to easily implement template matchings also at syntactic level; i.e., matching at RDF resource level.
- The transport layer (work package 4) is expected to communicate with the space in terms of graphs (set of RDF triples). This desire concerns first of all the coordination API of work package 3, which specifies the way external parties interact with the space. This interaction is independent of the way the semantic data is managed internally and separated by use of the API. Nonetheless, even if external clients provide the data in form of graphs, these can easily be parsed and treated by the space as triples.
- The use case work packages work package 8a [12] and work package 8b [15] could not provide any input that would influence the decision process. Like the Web service interaction patterns in work package 4, the applications resulting from the use cases are separated from the space by means of the coordination API and hence independent of the internal models.

Considering the feedback received from work package 3 and other related work packages and the fact that using a triple-based data model brings along considerable advantages with respect to distribution, replication and indexing strategies—which are to be elaborated in a second phase of the project—the partners decided to use the first approach. The advantages are mainly related to the fact that processing single triples is envisioned to be easier than addressing whole graphs. Single triple fields already reveal some limited knowledge about the content, while graphs need to be interpreted as a whole and by use of external RDF engines. Moreover, having three-fielded tuples as core data model does not disable the usage of RDF graphs as logical container.

However, the suitability of the triple-based model for handling more expressive knowledge representation languages needs to be approached in more depth in the future. The chosen model was estimated to be less appropriate for this purpose than a graph-based representation. As the focus of TripCom is primarily set on the usage of RDF and RDFS the extensibility of the model with respect to higher-order formalisms is considered as being of lower priority.

4.2 TripCom tuple model

The analysis of the alternative conceptual models revealed the benefits of using a triple-based flat tuple model for representing semantic data within a space. Nevertheless this model needs to be further specified in order to provide a feasible basis for the upcoming implementation of the triple space.

The first question we need to answer is related to the way tuples are *identified* within a tuplespace. One can apply a number of URIs to a given triple/tuple.

There is the URI which identifies the tuple in the space, as much as an URL with appended XPath expression could identify an RDF statement in an online RDF document. This URI exists outside of the tuple as much as the URL of an RDF statement is not part of the statement (it is something other than the reification of the statement). This URI can be the identifier of the tuple allocated by the system in modeling the space through a space ontology (cf. Deliverable D2.2 [31]). If a tuple is removed from and reinserted in the space it follows that it is a new instance of the class `Tuple` in the ontology and is allocated a new URI.

This "external" URI is useful in terms of the REST communication model. We can PUT or GET triples or graphs into or from certain URIs based on this form of identification, hence enabling RESTfulness in the space.

Given the triple-based model of tuples we agreed upon, one can imagine a second type of identifier in relation to semantic tuples, i.e. the graph URI which is associated to each RDF statement.[2]In this context we need to decide upon the basic structure of a tuple in the semantic tuplespace, the natural alternatives being triples and quads. In the former case a tuple would fundamentally contain three fields, corresponding to the subject, the predicate and the object of an RDF statement. In the latter the tuple would include an additional fourth field referring to the ID of the graph the statement is part of. The question is whether this ID should be a part of the tuple, or like the other identifier, should rather exist outside of the tuple structure. An alternative to the four-field tuple type would be to express triple membership in a named graph through another triple:

```
<<s, p, o>, ts:belongsTo, id>
```

The `belongsTo` property would be defined in the tuplespace ontology and used to attach triples to named graphs. A number of arguments can be made for this alternative:

- given that every RDF statement would be represented in the space in form of a triple, two subsequent insertions of the same statement would not cause the creation of duplicate statements in the space. This behavior is compliant to the RDF semantics, while in classical Linda systems tuples with the same contents may co-exist in a space. However two quads of the form differing in the ID field are in terms of their tuple structure different and technically would not be merged in a space. By separating triple content from identifiers, we preserve the semantics of matching from both a Linda and RDF perspective.

[2]A graph is defined as a (non-empty) set of RDF statements. If a statement is not an integral part of a graph, then we can consider the graph containing solely this statement instead and use an ID for this purpose.

- The usage of quads raises further questions with respect to the semantics of Linda primitives. Consider the case in which a quad is removed from a space and inserted it into a new one while preserving the same graph identifier. While we can allow graphs to be split across spaces, it seems more likely that statements will be removed from one graph and added to another, instead of shifting graph fragments across spaces. Again by separating triple content from IDs, we preserve the semantics of triples being associated to named graphs, rather than named graphs being part of the triple.

- By representing graphs as first-class objects in the tuplespace ontology we allow the possibility to talk about them, i.e. attach metadata to them. This is not possible, or at least less intuitive when a graph (or its ID) is conceptually an integral part of the tuple structure. In this case we would attach metadata to tuples, and could not differentiate between metadata applying to an RDF statement, or to the graph(s) containing it.

- By having graphs as part of the tuple structure we force the coordination language to include support for graphs as part of the coordination operations. If graphs are however part of the tuplespace ontology, we create and modify graphs solely at the level of this ontology and do not need to extend the coordination language with support for graphs and statements.

The triple associating an RDF statement to a particular named graph (i.e. using the aforementioned `belongsTo` relationship) can be structured in two different ways:

- Using a *nested* tuple. In this case the tuple containing the graph ID points to an underlying tuple containing the subject, the predicate and the object of the corresponding statement. This means the tuple has the form

  ```
  < <s, p, o>, belongsTo, graphId>
  ```

  Nested tuples are not easy to handle in terms of matching, since worst case they might require arbitrarily complex matching procedures. The basic problem is whether one applies a formal on the nested tuple or whether one introduces nested templates which are applied to the content of the nested tuples. Therefore we decided to use a second approach to this issue.

- Using a *statement identifier*. Since every RDF statement is associated to a URI in the tuplespace ontology, we could express the relationship between statements and graphs with the help of this identifier as follows:

  ```
  < URI of the <s, p, o> as defined in the TS ontology, belongsTo,
    URI of the named graph as defined in the TS ontology>
  ```

  The URI specified for each statement in the tuplespace ontology could be returned as a parameter by the `out` operation, through which the statement has been inserted into the tuplespace. The client can then associate the statement

to a particular graph using the tuple identifier. However, the coordination language needs to take into account additional methods for handling these URIs for matching purposes. This issue was communicated to work package 3 in order to synchronize the work done in the two work packages.

To summarize, the tuple model foresees the usage a three-fielded flat tuples, which are identified uniquely at the level of the tuplespace ontology. Moreover, each tuple is associated to graph identifiers and to the identifier of the space it is contained in (cf. Chapter 5 for a description of the tuplespace model).

The model of a semantic tuple can be further refined in terms of the knowledge representation language whose semantics is relevant for processing the tuple content in relation with matchings. As the prospected system foresees a semantic matching behavior in addition to the classical Linda procedures, it might be useful to differentiate between tuples embedding RDF, OWL or WSML data. While this distinction does not affect the basic tuple model, which remains a set of three fields, it triggers the usage of particular matching procedures, and thus needs to be stored in the space. Again, the tuplespace ontology can provide the means for defining these various types of tuples. The coordination operations are implemented as methods on these tuple classes, so we would differentiate between matching algorithms to use by overriding the matching methods in the class.

Open Issues The simplicity of the tuple model is granted at the cost of storing additional information in the tuplespace ontology. While the benefits of this separation are clear, it also imposes new constraints on the way the metadata (itself formalized semantically using e.g., RDF and OWL) is handled by the space.

The first issue related to the removal of metadata statements and the semantics of this operation: it still needs to be decided whether to allow in

```
<tupleId, ts:belongsTo, graphId>
```

to remove a triple from a particular graph rather than from the space as a whole. Further on, we need to investigate whether the `belongsTo` property modifies the semantic of the Linda operation at all.

These questions will be clarified in collaboration with work package work package 3 during the implementation of the first prototype.

5 Tuplespace model

In this chapter we elaborate on the way tuplespaces are structured and organized, i.e. the tuplespace model.

5.1 Analysis of the approaches

Based on the analysis of the related work performed in Chapter 3 we provide in this section a discussion of possible approaches to the problem of structuring and organizing the tuplespace for triplespace computing. The chosen solutions will be described in more detail in Section 5.2.

Potentially relevant approaches are discussed according to the following categories: structuring of spaces, organization of data within the space, and visibility of spaces. In other words, we look at

1. how to model and define a space,
2. how to store information in a space together with annotations and mappings for the data, and
3. how to ensure visibility of data, or rather how to restrict the visibility of or access to data.

5.1.1 Structuring of spaces

With respect to the structure of the space there are two different possibilities to distinguish. On the one hand there is the single space approach, where the space abstracts from a single central server, as in the original Linda; or on the other the possibility for a multiple spaces structure that partitions one interaction space. According to the partitioning strategy we can further differentiate between flat multiple spaces and hierarchical or nested spaces with the intermediate tree-like structure. The table 5.1 outlines the advantages and disadvantages of the four possibilities.

The single space approach is not adequate for an infrastructure that aims at a global information space due for performance and scalability reasons. On the other hand, the nested and interlinked approach, in particular when allowing overlapping spaces, necessitates more complex data management mechanisms. Consistency, and in particular updates over multiple spaces, implies a high processing overhead. The intermediate approaches of multiple flat spaces, or tree-like spaces seem to be more

	Pros	**Cons**
Single	Simplified implementation, simple access through single access point	Single point of failure, bad scalability
Flat	Distribution of data possible, better scalability	No structuring, no dependencies and relationships of spaces
Nested	High flexibility in structuring spaces and data	Difficulties in updating, high number of dependencies
Tree	Allows layering and limited nesting of spaces	Limited interlinking and dependencies

Table 5.1: Discussion of space structures

reasonable for the goals of triplespace computing. The advantage of a tree-like space organization is the possibility to partially merge a number of spaces within a parent space, while the flat management provides a simple means of organization without loosing the possibility to group communication partners.

5.1.2 Organization of data

With respect to the organization of data within spaces there are two aspects to discuss. The first one is related to the use of multiple spaces for administrative and application data, i.e., having for every data space a virtual space for the data published by users and separate spaces for administrative or meta-data. The second aspect to look at concerns the use of local scopes in addition to the installation of virtual spaces.

Multiple spaces: PageSpace for example use multiple spaces as well: one for the application data, and one for rules and service information. CSpaces on the other hand (cf. also Section 3.2) even applies seven distinct spaces to store all the information; amongst others for domain theory, security and trust, metadata, instance data, etc. Also the TSC project suggests the use of administrative spaces in order to clearly separate the user data from the internal management data.

An alternative to the usage of multiple spaces would be the use of administrative data graphs, mapping rule graphs and so on. The advantage of this approach is the simplified space model, as all data concerned with a given space is stored in the same data container. A major drawback however is the fact that eventually all data would most likely be treated the same way with respect to access control, distribution and replication. However, it is obvious that internal management data, security and trust information or mapping rules do neither have the same users nor do they expect the same handling. Having multiple distinct spaces provides a natural means to tailor the manipulation procedures.

Scopes: Spaces provide closed, persistent containers for information. Once the data is published in a given space, it is part of that space until its removal. Additionally, Semantic Web Spaces propose a means to access data of multiple spaces temporally and locally to a given node. In particular when refraining from the nested and overlapping space approach introduced above, this could provide interesting means to install more elaborated interaction channels. The idea is that a given user can locally gather information of various spaces and combine it temporally within a private scope; finally the user can query information over that scope or write information to the scope which is not reflected back to the original spaces. The advantage is obviously the added container (communication channel) that allows for interaction with an arbitrary set of triples, while the disadvantage is clearly the added complexity and the required overhead in defining and managing such scopes.

5.1.3 Visibility of spaces

The last issue to be discussed within this section is the visibility or accessability of spaces. The World Wide Web is a global information space built upon the Internet and many intranet. The former are generally public to all users, while the latter have a clearly restricted visibility. Moreover access control through password restrictions provides means to filter the number and type of users that are allowed to read a particular file on the net.

Similarly triplespaces need to provide means to restrict the access to information. CSpaces, analogue to the Internet-intranet solution, suggest so-called Individual CSpaces and Shared CSpaces.

In a more generalized way it could be expected that for every space the corresponding access rights can be defined. TSC provides means to define users, roles and access rights per role that can be assigned to every space. In that way the type of user that is allowed to read from, write to or both a space can be defined on an individualized basis. This approach is more flexible than the separation in public and private, while not refraining from the idea. Locally a user can define a space that only he/she has access to while at the same time being part of a global space that is shared with other users.

5.2 TripCom tuplespace model

In this section we specify the model used to structure and organize tuplespaces following the results of the analysis carried on in the previous section.

A tuplespace is a virtual set of tuples. Tuples are elementary data units, which encapsulate RDF statements of the type

```
<subject, predicate, object>
```

They can be identified using URIs and grouped in RDF graphs.

A tuplespace can be divided into virtual subspaces and physically partitioned across distributed kernels. Every space can be addressed using a URI identifier, which is created and defined in the tuplespace ontology. Just as in the case of individual tuples, this URI is useful in terms of the REST communication model. The meronymic structure (being a whole of parts) of a space with respect to the underlying subspaces is defined as follows: every space may contain multiple (sub-)spaces, while it can be contained in at most one super-space. Further on, tuples are associated to a single space. Hence, we do not allow overlapping spaces, both with respect to tuples and spaces.

Scopes are temporary tuple containers whose tuples can be taken from different subspaces. Unlike subspaces, which form the virtual structure of the tuplespace, scopes can be created individually by clients based on specified filters and are only available if shared with other clients. They can be given different semantics, e.g. they could be seen as an alternative view of the structure of the tuplespace from a particular client, or as a temporary copy of some tuples for retrieval by a client – in the latter case one would not allow insertion operations, and deletion operations would apply to the scope and not to the tuplespace as a whole. We note a last difference between spaces and their temporary counterparts: while spaces are defined to be non-overlapping, the notion of scopes does not impose this restriction. Consequently a tuple can be contained in one space and in a multitude of "scopes" (cf. [34]).

In addition to this classification, we differentiate between spaces storing application and administrative data, as a result of our analysis.

Note that the model described in this section refers solely to the virtual structure of a tuplespace. Issues of distributions will be approached in detail in subsequent tasks in this work package.

Open Issues There are several open issues with respect to the tuplespace model. The first refers to the possibility of storing tuples with identical content in the same space. While this feature is allowed in traditional Linda systems, it contradicts with the semantics of RDF, which defines equality on the basis of the contents (subject, predicate and object URIs are identical). Further on, a final decision with respect to the visibility of spaces, in particular the granularity of the access rights grants and the interplay between the rights management system and the tree-like tuplespace organization needs further research and will be addressed in work package 5.

6 Implementation outline

The task for initial implementation of the RDF triple semantics in tuples is part of the joint implementation effort of work package 1, work package 2 and work package 3 with a contribution of work package 6 to develop an early prototype of a triple space. The current prototype representation of semantic data in tuples addresses the results of the specifications of:

- T1.2 Specification of storage model and architecture based RDF stores part of D1.2 [37].
- T2.1 Specification of representation of RDF triple semantics in tuples described in section 4 in this document.
- T6.2 Identification of all components of the architecture including requirements and responsibilities of the components as part of D6.2.

There are several features which will not be available in the first prototype because overlapping with ongoing or yet not started tasks:

- Security and trust model in the triple space.
- Distribution of the triple space data over multiple nodes.
- Representation of the triple space through an ontology.

6.1 High-level objectives

The tuple model is regarded as the binding component between the Triple Store adapter [17], responsible to permanently persist the stored information, XVSM system [17] used as integrating space middleware infrastructure and the triple space API implementation that is the entry point and the visible front-end of the triple space prototype. The XVSM system interfaces provides the support of different logical coordination spaces refered to as containers. Every container could be identified by name and is capable to persist set of abstract data structures called Entry. The tuple space model implementation wraps the XVSM containers (spaces) and provides adequate support for the transparent mapping between the RDF types and the XVSM data types. Hence, the tuple model ensures a consistent mechanism for the communication of semantic data and its meta-data between the different components described by the Triple Space Reference Architecture [17].

6.2 Features of the implementation

The prototype representation of semantic data in tuples implements the conceptual model specified in Chapter 5 and introduces several optimizations to increase the efficiency of the triple space with respect to the used concrete systems. The XVSM named containers are used to logically isolate the tuples into different spaces. The method signatures of the in/out/read operations specified by the TS API interface [16] requires the maintenance of tuple function of 5 elements - subject, predicate,

object, graph name and space identifier. A new isomorphic structure to that proposed by the conceptual model is used for the implementation to lower the complexity of the operations over the space and increases the scalability of the system.

```
<subject, predicated, object, graph, space>
```

An alternative approach is to use nested tuples in the form of:

```
< <subject, predicate, object>, belongsTo, <graph, space> >
```

However the XVSM platform at the present time does not support nested tuples and the Linda template matching works only on the first level.

A strict implementation of the tuples and tuplespace models using internal identifiers would require higher complexity of operations to insert or remove data from the model in order to ensure consistency between the data and its related meta-data triples. The isomorphic structure presented above eliminates the generation of internal system identifiers and capsulate the data and its metadata. Thus, the problems related to the synchronization between data and its metadata, scope and the semantics of the internally generated identifiers are resolved. Further on, the usage of atomic tuples guarantee consistency with the semantics of Linda, where the "in" or "out" data operations is not expected to modify the internal structure of the space. The implemented tuple model is regarded as equivalent or isomorphic to the proposed conceptual model. Despite the efficiency and simplicity of the approach there is also disadvantage related to its flexibility: it is no longer possible to define arbitrary types of meta-information about the triples. That problem is not resolved in the first prototype implementation, because it reuses only the graph and space concepts from the Triple Space Ontology, but it is addressed in the deliverable Specification of Triple Space Ontology in Section 4 [31], where a more general and flexible solution is suggested and would be implemented in the future.

Another specific feature of the TripCom semantic tuple model implementation is the handling of the RDF blank nodes. According the named graph definition proposed by Carroll the blank nodes could not be shared across different RDF graphs [6]. The TripCom tuple model extends the understanding of a different RDF graphs also with different spaces. The implementation prefixes the internal blank node identifier with randomly generated namespace to prevent a possible incorrect usage.

Lastly, the tuple model implementation acts also as an adapter between the supported XVSM and RDF Java types. At the present time XVSM implementation supports only integer and UTF-8 string primitive types. All RDF java types are persisted as UTF-8 strings, where a single non-printable character is added in the beginning to denote the type of the value. The u0001, u0002 and u0003 non-printable characters are used to mark URI, blank node and literal types.

The tuple model implementation is released under LGPL license [1] and is publicly available at the TripCom project hosted on SourceForge [2]. Further related information could be found in D3.1 deliverable [16].

[1] http://www.gnu.org/licenses/lgpl.html
[2] http://sourceforge.net/projects/ordi

7 Summary and outlook

In this deliverable we presented the TripCom approach to representing and organizing semantic data, in particular RDF, within tuplespaces. First we discussed the requirements of a semantics-aware middleware and looked at the necessary extensions and adaptations of the original tuple and tuplespace models with respect to the representation of formal knowledge and the way this knowledge is published and exchanged on the Semantic Web.

The triple space model extends Linda mainly in what concerns the representation of semantic data in form of RDF triples and graphs. It was necessary to revise the syntax and semantics of tuples to reflect the fact that RDF triples are identifiable resources that presented nested and interlinked knowledge. Further on, the tuplespace model needed to be considered in order to cope with the ways semantic data is typically organized in semantic applications and to make triple spaces scalable on Web-scale (in contrast to traditional approaches that rather focused on corporate and thus small-scale solutions).

These considerations have been implemented in a first TripCom prototype, which is outlined in Chapter 6 of this deliverable.

With this respect we expect additional support from ontology-driven space management, one major assets compared to conventional tuplespace installations. In the remaining tasks of this work package, we will primarily concentrate on the development of tuplespace models which enable scalability at Web scale. While the heterogeneity problem is solved by the support for Semantic Web tools and dynamism is implicitly addressed by the inherited features of space-based computing, the scalability issue is still only marginally touched. We will therefore investigate mechanisms to tackle the significant challenges of distribution in large scale systems like the World Wide Web, Grid or pervasive computing environments. Semantic clustering of data, organization of spaces according to the internal structures of data and the joint usage of local and global spaces are possible starting points for future improvements to the existing semantic tuplespaces. Some of these ideas were already materialized by use of the triple space ontology in the parallel task 2.2, but not yet implemented. We thus expect that upcoming work will take up these ideas, and that solutions for the distribution and scalability issues will be developed around them.

Moreover, the future of the Semantic Web is seen in the integration of rules languages with the currently available W3C recommendations RDF(S) and OWL. Such complex knowledge representation formalisms and the associated sophisticated reasoning services they enable are still missing in triple space computing. These issues will be explicitly addressed in tasks 2.5 and 2.7.

References

[1] M. Bonifacio, P. Bouquet, and P. Traverso. Enabling Distributed Knowledge Management: Managerial and Technological Implications. *Novatica and Informatik/Informatique*, 3(1), 2002.

[2] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, February 2004.

[3] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Int'l Semantic Web Conf.*, pages 54–68, June 2002.

[4] G. Cabri, L. Leonardi, and F. Zambonelli. MARS: a programmable coordination architecture for mobile agents. *IEEE Internet Computing*, 4(4):26–35, 2000.

[5] N. Carriero, D. Gelernter, and L. Zuck. Bauhaus-linda. In *Workshop on Languages and Models for Coordination (ECOOP)*, July 1994.

[6] J.J. Carroll, Ch. Bizer, P. Hayes, and P. Stickler. Named Graphs. *Journal of Web Semantics*, 3(4), 2005.

[7] R. Chinnici, J.-J. Moreau, A. Rymanh, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Candidate Recommendation, March 2006.

[8] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, March 2001.

[9] P. Ciancarini. PoliS: A Programming Model for Multiple Tuple Spaces. In *6th IEEE Symposium on Software Specification*, 1991.

[10] P. Ciancarini, A. Knoche, R. Tolksdorf, and F. Vitali. PageSpace: An Architecture to Coordinate Distributed Applications on the Web. *Computer Networks and ISDN Systems*, 28(7-11):941–952, 1996.

[11] P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, and A. Knoche. Coordinating Multiagent Applications on the WWW: A Reference Architecture. *IEEE Transactions on Software Engineering (Special Issue: Mobility and Network Aware Computing)*, 24(5):362–375, May 1998.

[12] David de Francisco Marcos et al. TripCom Requirements Analysis and Architecture Profile for EAI Applications. TripCom Deliverable D8A.1, March 2007.

[13] R. De Nicola, G. L. Ferrari, and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

[14] e. Kühn. *Virtual Shared Memory for Distributed Architectures*. Nova Science Publisher, 2001.

[15] Dario Cerizza et al. State of the art and requirements analysis for sharing health data in the triplespace. TripCom Deliverable D8B.1, March 2007.

[16] L. Nixon et al. Specification and Implementation of a semantic Linda model. TripCom Deliverable D3.1, March 2007.

[17] M. Murth et al. Triple Space Reference Architecture. TripCom Deliverable D6.2, March 2007.

[18] D. Fensel. Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information. In *IFIP Int'l Conf. on Intelligence in Communication Systems*, pages 43–53, 2004.

[19] D. Fensel, R. Krummenacher, O. Shafiq, e. Kühn, J. Riemer, Y. Ding, and B. Draxler. TSC - Triple Space Computing. *e&i Elektrotechnik und Informationstechnik*, 124(1/2), Febuary 2007.

[20] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces: Principles, Patterns, and Practice*. Jini Technology Series. Addison-Wesley, 1995.

[21] D. Gelernter and N. Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2):97–107, 1992.

[22] P. Groot, P. Hitzler, I. Horrocks, B. Motik, J.Z. Pan H. Stuckenschmidt, D. Turi, and H. Wache. Methods for approximate reasoning. Knowledge Web Deliverable D2.1.2, January 2005.

[23] V. Haarslev and R. Möller. RACER System Description. In *Int'l Joint Conference on Automated Reasoning*, pages 701–705, June 2001.

[24] A. Harth, M. Magni, and St. Decker. Scalable Distributed RDF Storage Infrastructure. DERI Líon Deliverable 1.02, June 2005.

[25] P. Hayes and B. McBride. RDF Semantics. W3C Recommendation, February 2004.

[26] S. Hupfer. Melinda: Linda with Multiple Tuple Spaces. Technical Report 766, Department of Computer Science, Yale University, February 1990.

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

[28] L. Kagal, J. Undercoffer, A. Joshi, and T. Finin. Vigil: Enforcing Security in Ubiquitous Environments. In *3rd Grace Hopper Celebration of Women in Computing*, September 2000.

[29] D. Khushraj, O. Lassila, and T.W. Finin. sTuples: Semantic Tuple Spaces. In *1st Ann. Int'l Conf. on Mobile and Ubiquitous Systems*, pages 268–277, August 2004.

[30] R. Krummenacher, M. Hepp, A. Polleres, Ch. Bussler, and D. Fensel. WWW or What is Wrong is with Web Services. In *3rd European Conf. on Web Services*, pages 235–243, November 2005.

[31] R. Krummenacher, E. Simperl, V. Momtchev, L. Nixon, and O. Shafiq. Specification of the Triple Space Ontology. TripCom Deliverable D2.2, March 2007.

[32] R.M. MacGregor and I.-Y. Ko. Representing Contextualized Data using Semantic Web Tools. In *1st Int'l Workshop on Practical and Scalable Semantic Systems*, 2003.

[33] F. Martin-Recuerda. Towards CSpaces: A New Perspective for the Semantic Web. In *1st Int'l IFIP/WG12.5 Working Conf. on Industrial Applications of Semantic Web*, August 2005.

[34] I. Merrick and A. Wood. Coordination with Scopes. In *ACM Symposium on Applied Computing*, pages 210–217, March 2000.

[35] N.H. Minsky and J. Leichter. Law-Governed Linda as a Coordination Model. In *Workshop on Languages and Models for Coordination (ECOOP)*, August 1995.

[36] N.H. Minsky, Y.M. Minsky, and V. Ungureanu. Making Tuple Spaces Safer for Heterogeneous Distributed Systems. In *ACM Symposium on Applied Computing*, March 2000.

[37] V. Momtchev and A. Kiryakov. Specification of the Store Architecture and Interfaces. TripCom Deliverable D1.2, September 2006.

[38] A. Omicini and F. Zambonelli. Coordination for Internet Application Development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, 1999.

[39] E. Prud'hommeaux and A. Seaborne (eds.). SPARQL Query Language for RDF. W3C Working Draft, October 2006.

[40] B. Sapkota, V. Momtchev, J. Saarela, and S. Groppe. State of the Art and Requirements Analysis. TripCom Deliverable D1.1, June 2006.

[41] St. Schlobach and R. Cornet. Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In *18th Int'l Joint Conference on Artificial Intelligence*, pages 355–362, August 2003.

[42] G. Suryanarayana and R.N. Taylor. A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications. Technical Report UCI-ISR-04-6, Institute for Software Research, University of California, Irvine, July 2004.

[43] G. Sutcliffe and J. Pinakis. Prolog-Linda – An Embedding of Linda in muProlog. In *4th Australian Conf. on Artificial Intelligence*, pages 331–340, 1990.

[44] Geoff Sutcliffe and James Pinakis. Prolog-D-Linda: An Embedding of Linda in SICStus Prolog. Technical Report 91/7, Department of Computer Science, University of Western Australia, 1991.

[45] R. Tolksdorf, F. Liebsch, and D. Minh Nguyen. XMLSpaces.NET: An Extensible Tuplespace as XML Middleware. In *2nd Int'l Workshop on .NET Technologies*, May/June 2004.

[46] R. Tolksdorf, E. Paslaru Bontas, and L. Nixon. A Co-ordination Model for the Semantic Web. In *ACM Symposium on Applied Computing*, 2006.

[47] Robert Tolksdorf. Laura: A Coordination Language for Open Distributed Systems. In *13th IEEE Int'l Conf. on Distributed Computing Systems*, pages 39–46, 1993.

[48] Robert Tolksdorf and Dirk Glaubitz. Coordinating Web-based Systems with Documents in XMLSpaces. In *6th IFCIS Int'l Conf. on Cooperative Information Systems*, pages 356–370, 2001.

[49] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T spaces. *IBM Systems Journal*, 37(3):454–474, 1998.