TripCom

Triple Space Communication

FP6 – 027324

Deliverable

D6.2

Triple Space Reference Architecture

Martin Murth
Gerson Joskowicz
eva Kühn
Dario Cerizza
Davide Cerri
David de Francisco
Alessandro Ghioni
Reto Krummenacher
Daniel Martin
Lyndon Nixon
Nuria Sanchez
Brahmananda Sapkota
Omair Shafiq
Daniel Wutke

Executive Summary

This deliverable defines the high level architecture of the Triple Space infrastructure. The high level goals and the use case requirements provide a list of features that drive the design of the architecture. The architecture definition itself is divided into four parts:

First, the logical architecture is described by giving an overview of the Triple Space components and their functionality. As a second step, it is defined how these components will be integrated using a space based middleware system. Third, each component is described in greater detail, and fourth, the deployment architecture of a Triple Space kernel is defined.

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**	6.2	**Title**	Triple Space Reference Architecture
Work Package	**Number**	6	**Title**	Triple Space Architecture and Component Integration

Date of Delivery	**Contractual**	M12	**Actual**	31-March-06
Status	Version 1.0		final ☒	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Martin Murth (TUW) Gerson Joskowicz (TUW) eva Kühn (TUW) Dario Cerizza (CEFRIEL) Davide Cerri (CEFRIEL) David de Francisco (TID) Alessandro Ghioni (CEFRIEL) Reto Krummenacher (LFUI) Daniel Martin (USTUTT) Lyndon Nixon (FUB) Nuria Sanchez (TID) Brahmananda Sapkota (NUIG) Omair Shafiq (UIBK) Daniel Wutke (USTUTT)			
Resp. Author	Martin Murth		**E-mail**	mm@complang.tuwien.ac.at
	Partner	TUW (Vienna University of Technology)	**Phone**	+44 (1) 58801-18517

Abstract (for dissemination)	This deliverable takes the high level goals and the requirements derived from the use cases and defines a high level architecture for the Triple Space infrastructure. A description of the single components, the employed integration technology, and the deployment of the Triple Space are described.
Keywords	Triple Space reference architecture, requirements, logical architecture, physical architecture, component integration, implementation plan

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
2006-12-15	1	Martin Murth	First draft structure
2006-12-19	2	Martin Murth	Added first content
2007-01-21	3	Martin Murth	Revised document structure
2007-01-21	4	Daniel Martin	Added draft of logical architecture
2007-02-09	5	Brahmananda Sapkota	Updated section 3.1
2007-02-12	6	Daniel Martin	Reworked implementation plan
2007-02-13	7	Brahmananda Sapkota	Added sections 3.3.10, 3.3.11
2007-02-13	8	Daniel Wutke	Modified figures
2007-02-13	9	Omair Shafiq	Updated definition of metadata manager in 3.1 logical architecture
2007-02-13	10	Omair Shafiq	Added section 3.3.7 Metadata Manager
2007-02-15	11	Vassil Momtchev	Added section 3.3.2 Triple Store Adapter
2007-02-15	12	David de Francisco	Updated section 2.2
2007-02-16	13	Martin Murth	Added section 3.2
2007-02-21	14	Nuria Sanchez	Added sections 3.3.6 Mediator Manager and 3.3.12 Web Service Registry
2007-02-28	15	Lyndon J. B. Nixon	Added introduction, TS API, Distr. and Txn. Mgr.
2007-02-22	16	Brahmananda Sapkota	Revised sections 3.3.10, 3.3.11
2007-02-22	17	Daniel Wutke	Cleanup of USTUTT sections
2007-02-26	18	Martin Murth	Added summary
2007-02-28	19	Brahmananda Sapkota	Refined sections 3.3.10, 3.3.11
2007-02-28	20	Lyndon J. B. Nixon	Content revision
2007-03-01	21	David de Francisco	Section 2.2 reviewed
2007-03-22	22	Daniel Martin	Final versions of contributions
2007-03-18	23	Brahmananda Sapkota	Implemented Review Comments on Sections assigned to NUIG
2007-03-19	24	Geri Joskowicz	Added section about distribution and scalability
2007-03-20	25	Martin Murth	Implemented review comments
2007-03-23	26	Lyndon J. B. Nixon	Revised sections on Txn. and Distr. Manager
2007-03-23	27	Martin Murth	Finalised document revision

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

API	Application Programming Interface
DCM	Digital Contents Management
DAM	Digital Asset Management
DoW	Description of Work, TripCom Annex I.
OWL	Web Ontology Language
OWL-S	Semantic Markup for Web Services
RDF	Resource Description Framework
RDFS	RDF Schema
TripCom	Triple Space Communication
TSC	Triple Space Computing
TS API	Triple Space API
UC	Use Case
UML	Unified Modelling Language
W3C	World Wide Web Consortium
WP	Work Package
WS	Web Service
WSDL	Web Service Description Language
WSML	Web Service Modelling Language
WSMT	Web Service Modelling Toolkit
WSMX	Web Service Execution Environment
XML	Extensible Mark-up Language

1 Introduction

In order to make Triple Space a reality, it is necessary to define precisely what is Triple Space, i.e. out of which components will it be built, what will be the functionality of each component, with which components will each component exchange data and how. This deliverable is the first step in making that definition, termed Reference Architecture. Subsequently, the technical realization of each component will be decided upon (building from scratch, extending existing technology) and the reference architecture model and component interfaces (API) will be fully specified.

In this deliverable, we refer back to the high level goals of TripCom (Section 1.1) and the first high level specifications as well as a Triple Space feature set derived from the use case requirements (Chapter 2). On this basis we have been able to form a Triple Space reference architecture, consisting of a logical architecture, integration with space middleware, a set of component descriptions and a deployment architecture (Chapter 3). The next steps are outlined in the implementation plan (Chapter 4).

1.1 High Level Goals

The TripCom project has the goal of developing a global Triple Space, in which semantic information (triples, based on formal knowledge models such as RDF) can be distributed Web-wide and accessed in a coordinated manner by services. In taking this approach, we achieve global knowledge access by services in a loosely coupled manner, demonstrating de-coupling in time, space, and reference. Triple Space can become the global enabler of loosely coupled Semantic Web services, a coordination middleware for the exchange of knowledge between machines.

To meet this ambitious goal, Triple Space will be based on the following principles:

- Persistent publication and distribution of semantic data to achieve Web scale
- Data retrieval by semantic matching based on an extended Linda coordination model [20] to provide for time-, space-, and reference-wise de-coupling
- Mediation of semantic data to integrate heterogeneous services and applications
- Use of appropriate security mechanisms to guarantee confidentiality and trustworthiness of data
- Use of Web and Semantic Web technologies for integration with existing systems
- Application of Triple Space technology in industrial scenarios for validating scalability and robustness

1.2 Definitions

Throughout this document we use the terms listed below to describe the concepts of our architecture. We tried to create a clean separation between terms from the physical world (e.g. servers, network addresses, etc.) and terms from the concepts of the Triple Space - which we refer to as the logical world in this document. A detailed description of the defined terms is given in Chapter 3.

Logical entities:

Triple Space: A Triple Space is a set of semantic data that is identified by the same identifier, i.e. the name of the Triple Space. It is managed by a set of Triple Space nodes and can be accessed via the Triple Space API primitives.

Sub-Space: A sub-space is a subset of data in a Triple Space that is identified by a specific context. A sub-space complies to the definition of a Triple Space and can have sub-spaces itself. Accordingly, a sub-space may also be distributed over multiple nodes.

Node: A node is an instance of a kernel and therefore exposes the same API. A node is managed under a single authority and is identified by a unique logical identifier. In compliance with the URI syntax[1], this identifier consists of a scheme ('xvsm'), an authority (the DNS name of a kernel) and a path that identifies a node relative to its corresponding kernel.

For example, the full name for a node "Milano" on kernel "tcs1.tripcom.it" would be `xvsm://tcs1.tripcom.it/Milano` (Section 3.4).

Physical entities:

Kernel: A Triple Space kernel is a single physical implementation consisting of all necessary components implementing the Triple Space API. A kernel is deployed on a physical infrastructure (Section 3.4) and is addressable using a specific physical network address like a single physical machine or a single physical cluster of machines. One kernel can host an arbitrary number of nodes.

Kernel Component: A kernel component is part of a kernel, provides a specific functionality and takes over a certain responsibility in the kernel's architecture (e.g. components for security, distribution, mediation, and persistence; Section 3.1).

[1] see http://www.gbiv.com/protocols/uri/rfc/rfc3986.html#components

2 Triple Space High Level Specification and Features

This chapter describes the main concepts related to the Triple Space infrastructure. The first section presents a high level specification of the Triple Space infrastructure and the second section focuses on the features that such an infrastructure has to provide to external services and applications.

2.1 High Level Specification

As a first step, the general scope of Triple Space has to be defined and specific assumptions need to be validated before defining the architecture.

Triple Space is a distributed infrastructure without any central, single point of control. The Triple Space is physically implemented as a set of interacting kernels (built on top of peer-to-peer and Web technology). Each kernel acts under a different authority, i.e. there is no central authority controlling the various kernels (see Figure 2.1).

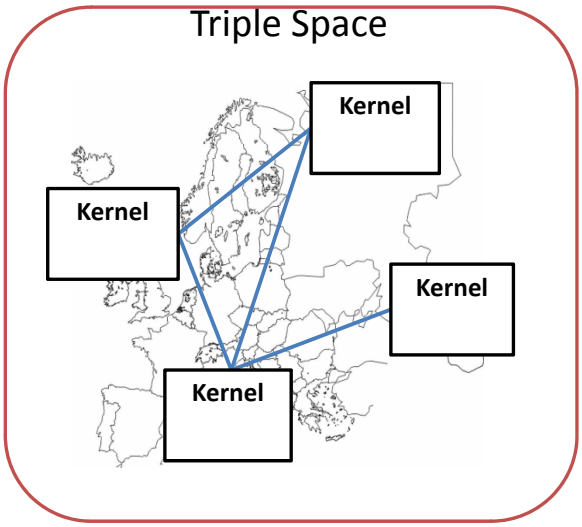

Figure 2.1: A Decentralized Triple Space Infrastructure

Each kernel exposes the Triple Space API [20]. This API embodies the paradigm of a global space which means that the client is able to transparently reach all information and services offered by all Triple Space kernels by connecting to any kernel (see Figure 2.2). It is not necessary to explicitly know (address) the kernel that manages the data to be accessed. Once connected to the global Triple Space using a particular kernel, the Triple Space infrastructure takes over routing of distributed requests.

The number of kernels and the presence of kernels are undefined at a specific point in time. As the Triple Space will be a highly distributed open system, there are no guarantees regarding kernel availability. Kernels can leave or join at any point in time, kernels can crash, etc. Consequently, vital information such as kernel's properties and states, parts of the Triple Space kernel topography, security principals and their authentication/authorization data, and runtime information must be distributed and replicated to ensure its availability.

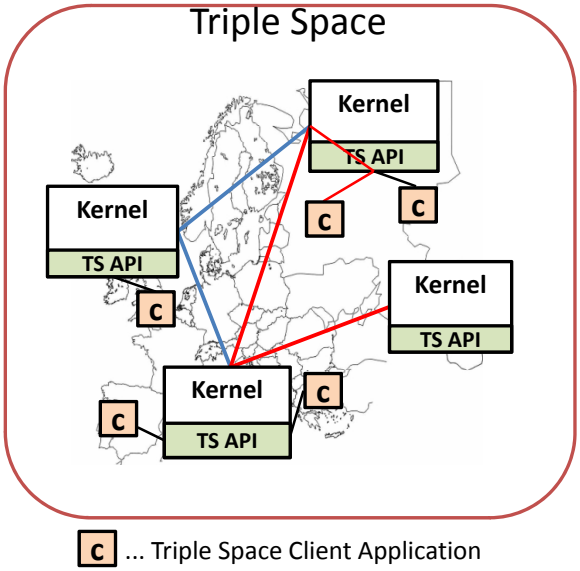

Figure 2.2: Accessing the Triple Space

2.2 Feature Set Derived from Use Case Requirements

This section reports on the set of features required to the Triple Space as described in the use case scenarios outlined in D8a.1 and D8b.1. Features are subdivided in some feature sets with regards to the main functionalities of the Triple Space. In this section, the Triple Space is considered as a whole black box capable of providing the functionalities of the Triple Space itself with also the functionalities of those external services eventually connected with the Triple Space (such as WSMX, mediation services, . . .).

Features Related to the Dimensionality of the Space

These features are related to the capability of the Triple Space in providing a distributed infrastructure that can support the European scale of the EPS (European Patient Summary) scenario.

- Support the number of healthcare authorities (10^4 nodes)[1]
- Support the number of users (caregivers, administrative staff) registered into the space (10^6 users)[2]
- Support the storing of the summaries of all the European citizens (10^{12} triples)[3]
- Support the concurrent access of 10^3 users per day on the same physical kernel and 10^6 users per day on the whole space
- Support for queries that get combined results from the whole space

[1]Each hospital, clinic or healthcare administrative center is intended to be a node of the space.
[2]Since the EPS is intended for sharing health data across healthcare authorities, only caregivers and administrative staff will be the users of such application.
[3]The triplespace have to manage the summaries of the 800 millions of European citizens. Each summary is about one thousand triples.

Features Related to Reliable Infrastructure

These features represent the capability of the Triple Space in providing a robust infrastructure capable of avoiding data losses and providing services in many critical situations.

- ACID Properties on transactions to avoid inconsistences when concurrent operations occur on the same data
- Fault Tolerance: the functionalities provided by the Triple Space have to be available also under critical situations, such as stress conditions or a failure of a physical component

Features Related to Data Interoperability

These features describe the capability of the Triple Space in bypassing data heterogeneity both at instance and schema levels.

- Mediation at schema level, in order to solve heterogeneity between different message format, that come from eHealth[4] and EAI standards
- Mediation at instance level, in order to solve heterogeneity between different terminologies[5], that come from eHealth standards

Features Related to Coordination Mechanisms

These features describes the capabilities of the Triple Space in coordinating external agents through interactions with the Triple Space.

- External applications can subscribe/unsubscribe when an access on specific data occurs in the space. For example, when a new entry is inserted in the summary of a citizen, the application of the citizen's general practitioner has to be notified of such new entry.
- Retrieval of matching triples from a query must be returned in an ordered way, where the parameter(s) for filtering the results have to be provided by the user
- Automatic Web service invocation from the Triple Space when an specified message arrives (event triggering).
- Orchestration of nested Web service calls in order to execute a business flow implemented with Web services.

Features Related to Security Access

These features aim to guarantee that only authorized users are able to access the sensitive data managed inside the Triple Space.

- Concept of authority

[4]See section 2.1.1 of D8b.1 for further details
[5]See section 2.1.2 of D8b.1 for further details

- The users[6] who can access the data in the EPS are defined and in charge by an authority
- Authentication of users
- Concept of role
- Roles are defined by an authority
- Users assume one or more roles
- Track any authorized or denied access to data, specifying also the user who performed the operation, the date-time of the operation and the data accessed or modified
- Authorization of accesses based on role-users and schema-instances. The permissions that could be managed are:
  - Read data
  - Add data
  - Remove data
  - Read or change the set of permissions on a specific schema or instance
  - Subscribe and remove subscription on access on specific data
  - Read the audit of a specific instance, that is: which users accessed the specific data, when and what kind of access it was (read, add, remove, change permissions, audit)
- Support for authority hierarchy
- Authorities can redefine roles by extending or renaming the roles inherited from the broader authority in the hierarchy
- External application should communicate with the Triple Space through secured communication channels (such as SSL, HTTPS, ...)
- Reputation based mechanisms which will check a user's credentials, allowing or denying a priori user acceptance in the system

[6]In the EPS scenario, the users are all the caregivers and administrative staff working for one of the healthcare authorities of Europe.

3 Triple Space Reference Architecture

This chapter presents the high level architecture of a Triple Space kernel from a logical and a deployment point of view and describes how the architecture components are integrated with a space based middleware system. The implementation tasks resulting thereof are linked to particular tasks of the TripCom Workplan (see Appendix A).

3.1 Logical Architecture

Figure 3.1 shows a logical view on a single Triple Space kernel (Section 1.2) with both, kernel-internal components and kernel-external clients and services that may be connected to it. From a birds-eye view on the architecture one can see two different areas. The upper area shows kernel-external entities that may connect to the kernel. The area below shows the components of a kernel itself, i.e. APIs and components implementing these APIs. The components that form a Triple Space kernel communicate over a kernel-internal bus system which is implemented using a space-based middleware system (Section 3.2.1). The integrating middleware allows all components that are drawn directly above or below to communicate with each other in a bus-like manner. The bus itself is depicted as a yellow arrow in the lower part of the figure.

APIs (blue boxes) describe logical groups of functionality through visible interfaces, hiding the actual implementation and are only accessible in a top-down manner; i.e. it is not allowed that a lower level entity accesses an entity above it.

Triple Space clients (green boxes) and *services* (green circles) are kernel external and interact with the Triple Space using the Triple Space API (defined by WP3 in [20]), the Web Service API (defined in WP4), and the Management API (defined by WP3 and WP5). *Mediation services* provide kernel-external transformation functionality used by the mediation manager 3.3.5. An *Orchestration engine* can be used to build higher-level functionality based on single Web services through service composition e.g. using a BPEL [5] engine.

Components (orange boxes with round corners) directly below an API implement the API's functionality. Components connected to the integrating middleware can only interact with other components over the bus - they are not allowed to interact directly. Kernel-internal components above the Triple Space API (e.g. the Web service invocation component) communicate using the Triple Space API in the same way as kernel-external entities do.

The following is an overview over the components that make up a single Triple Space kernel. Each of those components will be explained in greater detail in Section 3.3.

Triple Store Adapter: This component waits for requests to the triple store and uses the appropriate storage API to read and persistently store data. Every other component may use this one to store runtime data (e.g. a components internal state) into persistent storage.

Security Manager: This component verifies that requested operations do not violate the specified security policy. The security manager implements the policy decision point. The policy enforcement point is implemented in the virtual shared memory based bus system.

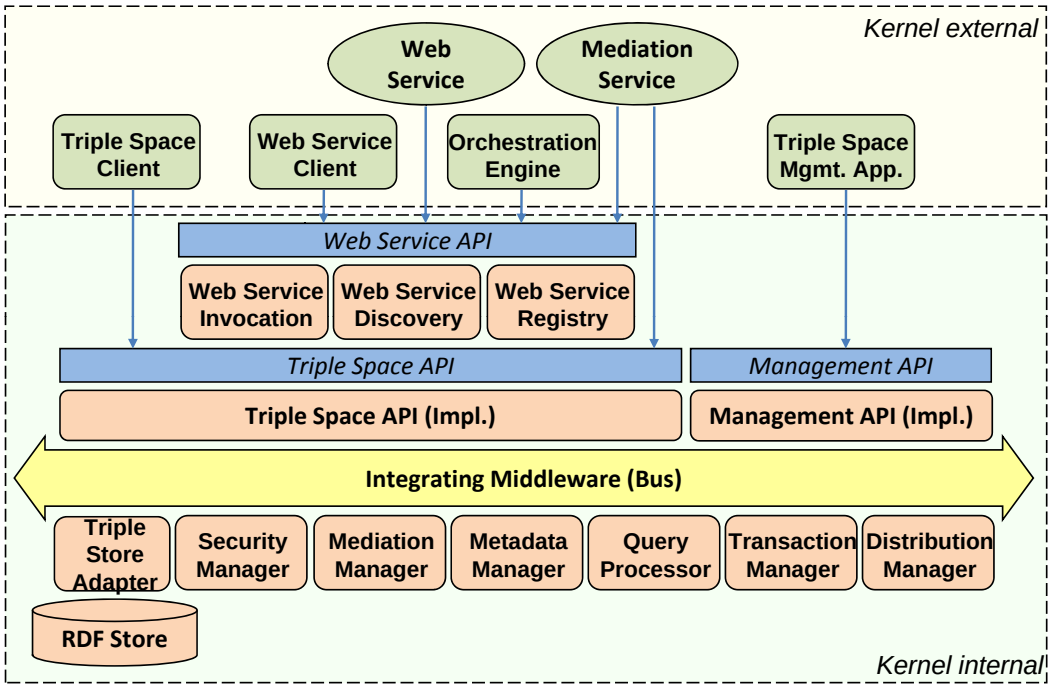

Figure 3.1: Triple Space Architecture – Logical View

Mediation Manager: This component is used to perform semantic data mediation on incoming and/or outgoing triples/graphs. Mediation can take place outside of the space implemented by a service or as part of the Triple Space itself using e.g. Abstract Mapping Language (AML) [24].

Metadata Manager: The Metadata Manager is responsible for the implementation of the functionality described by the Triple Space ontology. It acts as a reasoner to the TS Ontology that takes care of validation of ontology instance data across its schema and reasons to find out the information about data as required by users, e.g. calculating access statistics for triples, graphs, named graphs, logical sub spaces etc.

Query Processor: The query processor is responsible for decomposing a query to parts that are satisfiable by the local data store and to parts that must be forwarded to other kernels in order to fulfill the query in its entirety. This is achieved by working closely together with the distribution manager component.

Transaction Manager: This component manages local transactions. It may also act as a participant for distributed transactions. It even may act as a coordinator (2PC) if the distributed transaction was started on the kernel it belongs to. Part of this functionality may be achieved by working together with the distribution manager.

Distribution Manager: This component is responsible for the implementation of a distributed Triple Space, e.g. by forwarding queries and requests to kernels that may be able to partially satisfy them and forwarding of write requests to appropriate kernels as part of a semantic clustering of data.

Management API: This component will be required to set up initial data structures needed for communication between other Triple Space kernel components and to coordinate kernel components. It is responsible for bootstrapping the whole kernel process.

Web Service Invocation: The Web service invocation component is performing the actual interaction with an operation on a Web service. This mainly involves the translation of Web service interactions into Triple Space API operations to transport Web service data from the service requester to the service provider.

Web Service Discovery: The Web service discovery component is responsible for finding Web service descriptions that match a requesters goal.

Web Service Registry: Provides functionality to transform WS descriptions (e.g. in WSML) to RDF graphs and to store them in the Triple Space. This functionality will be implemented as part of TripCom, rather than in each client application.

Furthermore, there are the following kernel-external entities:

Triple Space Client: typical "Linda-style" clients which access the Triple Space using the Triple Space API directly.

Service: Web services hosted on top of the Triple Space; these services can be used by Web service clients.

Mediation Service: Mediation services can be connected to the Triple Space either via the Triple Space API or via the Web service API. They provide the functionality to map between concepts and ontologies.

Orchestration Engine: Optional external component that provides Web service composition functionality (e.g. a BPEL engine that uses the TripCom Web service binding to invoke activity implementations).

3.2 Component Integration with Space Middleware

The implementation of a Triple Space kernel has to be — like the Triple Space as a whole — robust, fault tolerant, and load balanced. Space technology proved to support these features in a number of projects and applications (e.g. [7, 14, 19, 25]) and consequently, we also employ a shared space middleware system to integrate the components within a kernel itself. More precisely, an XVSM (eXtensible Virtual Shared Memory) system (Section 3.2.2) will be used for realising data exchange and coordination of the kernel components.

3.2.1 Space Based Integration of Software Components

In a space based integration architecture, all components are autonomous and interact with other components by accessing and observing a common data space. Depending on the space middleware system, different operations can be used to manipulate shared data, all varying in the exposed interaction style and expressiveness.

Each component is clearly demarcated and characterized by a special function and responsibility. A lifecycle, defining who will create the component and when it will be created and terminated, has to be specified. A component may use its own resources (e.g. reference data stores) and some components will also communicate with the outside world through additional external interfaces and protocols (e.g. Web Service Invocation component).

A component interacts (coordinates) with other components using the shared space. The functionality and behavior of a space component can be described by a set of events and actions (cf. [17]). An *event* describes the kind of state or state change within the space which will trigger a specified action. For example:

- **Event "New Order Item"**: fired when a new order item is added to the shared space.

An *action* describes a task in terms of interactions with the shared space and other, external entities. For example:

- **Action "Process New Order Item"**:
  1. Remove new order item from space
  2. Process order item
  3. Write new invoice item to space
  4. Update log entry in space

The incoming events and the performed actions depict the start and endpoints of processing chains in the component. The entire set of events and actions of all space components reflects one or multiple workflows that are executed by the potentially distributed set of components.

3.2.2 XVSM - eXtensible Virtual Shared Memory

eXtensible Virtual Shared Memory (XVSM) [9] is an open space-based coordination architecture unifying and extending a number of principles and best practices from space-based computing architectures like Linda [12, 13], JavaSpaces [26], and Corso [7]. The functionality that has to be provided by an implementation to be XVSM compliant is defined in terms of a protocol specification (XVSM protocol).

XVSM Shared Data Containers

The basic building block in XVSM providing various entry points for extensions is the *XVSM shared data container*. It holds data shared by all agents connected to the space and provides a means for synchronized access to this data. A container has a unique identifier (in the form of a URI). It abstracts a shared coordination buffer and provides the basis for various coordination mechanisms. The data items held by a container are called entries. Entries are not restricted to any kind of data type but can be any kind of serialized data (e.g. nested tuples, XML data). Also references to other containers can be stored which allows for the construction of complex, interlinked container structures.

An XVSM runtime hosts one or more containers that can be located in the Internet via their URL. The abstract container defines a uniform interface to access the contained data: entries are inserted into a container via the operations *write(selector, entry)* and *shift(selector, entry)* and retrieved via *read(selector):entry* and *take(selector):entry*. *Selectors* control how and which entries are inserted and retrieved. Synchronization is based on a bounded buffer model, where blocking is symmetric: Both retrieval and insertion operations may block. Write inserts a new entry without affecting other values and blocks, if the container cannot accept the entry. Shift inserts a entry analogously to write, however it never blocks, but may displace another entry. Read returns a copy of an entry for the given selector and blocks, if there is no suitable entry available in the container. It does not change the container. A take is like read but also removes the returned entry from the container. A take operation may also block if no suitable entry is available. All operations take an optional timeout parameter which specifies how long an operation may block. A zero timeout is used to express non-blocking operations. The capacity of a container determines the maximum number of entries a container can hold. It is set at container creation and for a materialized container is an integer >= 1 or conceptually infinite. If the number of entries in a bounded container is equal to the container capacity, a container is said to be full. If not explicitly restricted, a container has infinite size.

Concrete container types inherit and do not extend the uniform interface but provide different coordination types which define the exact semantics of read, take, write and shift (see examples below). A programmer can choose from a rich set of generic container coordination types and select that one which best fits a given coordination problem. This frees the programmer from re-implementing recurring coordination patterns again and again on top of a too primitive and minimal coordination model.

Concurrent access to containers is managed by transactions. A sequence of operations executed within the same transaction context must be guaranteed to preserve ACID properties (atomic, consistent, isolated, durable).

Since there are no dependencies between entries in a single container, a container can be distributed, replicated, and cached on multiple nodes of a a distributed XVSM system. An end-to-end argument protocol [22] is employed to guarantee consistency of the distributed data.

In the following we present some coordination types supported by XVSM containers.

Examples for XVSM Coordination Types

Linda: Using the Linda coordination type provides an abstract generalization of a Linda-like tuplespace. A read requires a selector parameter of type *Template* and non-deterministically returns the copy of any matching entry, which must be *Matchable*. Template and Matchable are abstract and constrain that a Matchable must be able to match against a Template. If no matching entry exists in the container, read blocks; analogously for take. A write blocks if the container is full. A shift has no corresponding operation in Linda. It is like write, but non-deterministically displaces any value, if the container is full. Concretisations of this abstract container type can implement template matching based on various data and template models, such as Linda tuples and templates [12], Java objects and polymorphic template matching [11], XML doc-

uments and XML query-languages [28], and RDF graphs and RDF templates [10].

Fifo: The need for first-in-first-out coordination is indicated by the broad application of message queues [16]. Also GigaSpaces [15] optionally provide Fifo-ordering, but without blocking capabilities and without extensibility towards other coordination possibilities. An XVSM Fifo container is comparable to a queue and does not constrain the data model for entries. A read returns a copy of the first entry ("head of the queue") and blocks, if the container is empty. Take operates the same way but removes the first entry. A write appends an entry at the end of the queue and blocks if the container is full. Blocking write is a useful mechanism to realize back-pressure in communication (e.g. when very large amounts of warehouse data must be communicated [8]). A shift is like a write and writes also to the tail, but displaces the first entry if the container is full. No selectors are required with Fifo containers.

Also other coordination types such as a set, a map to hold key-value mappings, ordered and unordered collections, and last-in-first-out ordering can be provided under the same canonical interface. XVSM is not limited to these container types and can be extended by defining new ones.

3.3 Triple Space Kernel Components

In the following sections, we describe how each individual component of a Triple Space kernel integrates into the overall system. The set of component descriptions renders the complete behaviour of a Triple Space kernel and, thus, the behaviour of the entire Triple Space.

The component descriptions define the general functionality of a component, its coordination interface (i.e. how it interacts with other components via the integration bus), external interfaces, and the component's lifecycle. According to the current stage of the respective work packages, for some components there is already a precise description of incoming events and outgoing actions available. For others, a general description of the required interactions is given. An updated and more detailed description of the components internal structure and functionality will be defined in deliverable D6.3 "Platform API specification for interaction between all components".

3.3.1 Triple Space API

Functionality

The Triple Space API implementation is tasked with handling the requests expressed in terms of the Triple Space API received by the system from clients and services. It ensures the completion of each request, either terminating in failure or returning a response to the client/service, respecting the defined semantics of the API. It also coordinates between concurrent requests arriving at the kernel.

Coordination Interface

- Incoming events are Triple Space API operations. These are received by a Triple Space API interface directly from external clients / services or are being generated from other internal components as mappings from interactions taking place

either within the kernel (Triple Space Manager) or being instigated by external services (Web Service component stack).

The types of event in the API can be divided into the following: tuple insertion, tuple retrieval, tuplespace organization, transaction and notification

- Largely, the API implementation hands on each API request to the appropriate component in the appropriate form, using the integrating middleware as a coordination backbone. Its single internal processing task is to register each active request and to monitor its completion, either by establishing failure (and hence returning an error to the request originator) or acquiring a result. Generally, this is achieved by inserting messages into the integrating middleware and specifying a notification request for when a particular message is available. The messages will likely pass through a chain of components (or be handled concurrently), while the API implementation need only wait for a final "completed" or "failed" type message for that operation.

  For example, tuple insertion and retrieval will generate a message in the integrating middleware to be picked up by the metadata manager, which can determine the appropriate logical space which will take care of the operation. Likewise, tuplespace organization changes the logical structure of Triple Space which is updated in the metadata manager.

  Transactional operations are picked up by the Transaction Manager.

  Notifications and timeouts are special cases for the API implementation. Internally, active notifications and timeouts on operations are maintained and monitored. Notifications will trigger regular checks for matching triples and are only removed when an unsubscribe request is received. Timeouts will generate 'failure' messages on that operation once the time period is reached and no 'completed' or 'failed' message for that operation has been found in the integrating middleware. A timed out operation generates a 'timed out' message to the integrating middleware so that the component(s) working on that operation can cancel their activity.

Other Interfaces

- For the efficient operation of the API implementation, it may be necessary to make use of efficient storage to maintain the lists of active notifications and timeouts on operations, as well as to provide a data cache for recent or open (not yet completed) operations/data or a queue for client requests (if the amount of requests is exceeding the capacity of the kernel to handle them immediately).

- Triple Space implements a semantically enhanced Linda coordination model (see D3.1 [20]). While the defined API is stateless, a form of return channel to the client is necessary, either directly from the API implementation or over the Web Service stack, in order to return notifications and errors. It will be dependent on the protocol used and supported by the Triple Space API interface which lies between the API implementations and the "external" entities (clients/services). In the API implementation, each operation carries an internal identifier which must be associated at the interface with a client.

Lifecycle

The API implementation is a core part of the Triple Space kernel and is initialized with the kernel itself. It should offer persistent storage for its internal data (e.g. notifications, active timeouts, uncompleted operations) so that after a restart, the Triple Space kernel can continue to operate as at the time it stopped. In ordinary cases, only a system administrator should be able to create or destroy a kernel process.

3.3.2 Triple Store Adapter

Functionality

The Triple Store Adapter is an adapter to map the programming interfaces between the integrating middleware (XVSM platform) and the ORDI framework developed in WP1. It can also be regarded as domain specific model, part of the ORDI framework and built on top of the tripleset data model [6], to operate with the XVSM system trough the tuple space model. The Triple Store Adapter and the underlying infrastructure are responsible for providing the following functionalities:

- It ensures durable storage of RDF based data and the associated metadata saved in the integrating space middleware platform until explicitly removed by a user or system request.
- It evaluates RDF queries against the stored semantic data.
- It may support efficient reasoning capabilities over the stored information.

Coordination Interface

The Triple Store Adapter component is a service component to handle requests to insert, remove, and retrieve durable semantic data. It is responsible to process specific requests generated by other components. So far, there is no identified functionality offered by other components to be used by the Triple Store Adapter.

- Event: Semantic data inserted into the space: The event is raised whenever new semantic data is inserted into the space.

  Action: The Triple Store Adapter reads the newly inserted semantic data.

  Source: TS API or any component that is able to generate new semantic data.

- Event: Semantic data taken from the space: The event is raised whenever semantic data available from the space is taken by a specific component via specific query.

  Action: Remove the persisted data from the storage and invalidate the inference closure.

  Source: TS API.

- Event: Evaluate query request: The event is raised whenever a query evaluation request is inserted into the space.

Action: The adapter tries to recognize the query format and if processable executes it against the stored data and writes the retrieved results back to the space.

Source: TS API or any component being able to generate SPARQL queries.

The precise coordination between the Triple Store Adapter and the XVSM system is subject to further evaluation. Both systems provide functionality to store and retrieve information; however they are optimized for different types of tasks. The Triple Store Adapter is capable to provide efficient management of large amounts of durable data, whereas the XVSM system is able efficiently to handle large number of short lived transactions over limited sets of information. At the current state of the project we do net yet differentiate between these both types of information. An analysis of how to best integrate the internal space middleware with the Triple Store is subject to the next development phase of the project (Task 6.3., Evaluation, selection and definition of necessary extension of existing space middleware technologies).

Other Interfaces

The Triple Store Adapter will not offer any external service interfaces. In a later stage introduction of service interface for efficient data replication, may be required.

Lifecycle

The Triple Store Adapter is a core part of the Triple Space kernel and is initialized and terminated with the kernel itself.

3.3.3 Management API

Functionality

The purpose of the Management API implementation is to enable the initialization and maintenance of the internal data structures and processes of Triple Space for the kernel. It provides the implementation for the Management API which is in the layer above and offers functionality like kernel startup and shutdown plus management and administration related functionality like adding users, editing security policies for the kernel, prociding statistical data about the currently running kernel, etc.

Coordination Interface

Incoming events are either initialize or close. An initialization event from the system administrator starts the kernel process on the machine and sets up all necessary initializations, such as:

- acquiring the identifier for the kernel
- registering the kernel as a physical component in the Triple Space network
- starting the component implementing the Triple Space API
- initializing the network communication subsystem (making the kernel reachable to external clients at a particular address over certain protocols)

- providing the internal data structures which allow kernel components to create and manipulate internal data such as triples, tuplespaces and templates
- starting an integrating middleware process or finding and connecting to an existing instance
- providing a management API which allows for management functions, some of which may only be available to a kernel administrator and some may be more widely available (e.g. providing a machine readable description of the node to a client)

A close event ends all processes of the kernel, performing the necessary shut down steps such as deregistering the kernel on the network, saving internal data for recovery, handing off open tasks to other kernels or garbage collection.

Other Interfaces

- Some functionality of the Management API may be accessible through the Triple Space interface (to allow for remote access) but primarily will be locally available on a machine to the kernel administrator. This is also highly dependant on the local security policy and if this allows e.g. a kernel shutdown to be initiated from a remote.
- Some initialization parameters may be made available through data accessible in a local file or at a network address.

Lifecycle

The Triple Space management API implementation is a core part of the Triple Space kernel and is initialized with the kernel itself. In fact, it is the component which embodies the initialization method. In ordinary cases, only a system administrator should be able to start or stop a kernel.

3.3.4 Security Manager

Functionality

The Security Manager's main functionalities are authentication and authorization. The Security Manager implements the policy decision point of the kernel, whereas the policy enforcement point is implemented by the integrating middleware.

Coordination Interface

Regarding authentication, as the client tries to connect to a kernel, the kernel authenticates to it using an X.509 certificate within the TLS protocol. On the other side, the client can authenticate itself to the kernel using an assertion provided by an external Identity Provider service. Assertions can be expressed using SAML and can contain some attributes related, for instance, to roles that a user can have. The Security Manager checks the assertion and decides whether to trust it or not. If it is trusted, the Security Manager returns to the client an "authentication token", which can be used

by the client for following interactions with it by simply sending the token along with the requested operation on the space.

Regarding authorization, the Security Manager is the policy decision point. The Security Manager takes care of any incoming request, and validates it upon policies and access rights. Access control operates on a "security container" corresponding to the sub-space and thus security containers inherit the same hierarchy that sub-spaces have. Other components (e.g. the query processor) consider only requests that have been validated by the Security Manager and carry a proper authorization "mark". For this reason, the Security Manager would probably need a privileged access to the space: no other component will need to perform the same kind of access (e.g. marking a request as "authorized"), so it is better not to expose it to other components at all. This privileged access must be enforced by the integrating middleware.

Other Interfaces

The Security Manager may need to contact external authentication services (e.g. certification authorities for certificate revocation lists check).

Lifecycle

The Security Manager is a core component and its presence is always required: it starts as the TripCom kernel is started and finishes as the kernel is shut down. If no security is desired (i.e., everybody can do anything), the security manager must still exist, but it will be configured with an "always allow" security policy.

3.3.5 Mediation Manager

Functionality

In case two entities need to communicate and cannot do this directly due to different data formats (syntax) or different ontologies (semantic), a Triple Space component must be able to mediate among these entities and to manage the necessary mapping so that the communication can be carried out. The mapping between ontologies can consist of: mapping similar concepts among ontologies or transforming the instances belonging to an ontology to instances belonging to another ontology. Therefore, the Mediation Manager is in charge of consuming the events which need mediation and coordinating the process of performing necessary mappings in data level, interacting with other components in the architecture, and handling the results.

There are several issues to discuss in the following phases of the project: the way that the Triple Space detects the needed mediations, the kind of mediation required, and how it will be carried out.

Coordination Interface

- Event: Graph inserted into the space: The event is raised whenever a new graph is inserted in the Triple Space.

  Action: The mediation component sends a call to the Mediation Rule Repository (MRR) by means of Triple Space API to find the mapping rules that matches with the graph passed as parameter. In the MRR are stored, when a provider

publishing a service description in Triple Space, the transformation rules for mapping the ontologies used in the service description to ontologies based on standards commonly used. When the Mediation Manager receives the response to the call from MRR with the found transformation rules, it invokes the corresponding transformation service. Once the transformation service has been executed, the resulting graph (link to original graph) is stored in the Triple Store.

Source: Integration middleware transfers the event sent from Triple Space API to the Mediation Manager Component.

- Event: Query evaluation: The event is raised whenever a query is evaluated. This event is synchronous.

  Action: The mediation component sends a call to Mediation Rule Repository (MRR) by means of Triple Space API to find the mapping rules that matches with the graph passed as parameter. When the Mediation Manager receives the response to the call from MRR with the found transformation rules, it invokes the corresponding transformation service. Once the transformation service has been executed, the resulting graph is passed to the Query Processor Component.

  Source: Integration middleware receives a query evaluation from Query Processor Component and transfers this query by means of an event to Mediation Manager Component.

Other Interfaces

- Interfaces to Web Service Registry, when a graph is inserted in Triple Space and needs data mediation.
- Interfaces to Web Service Discovery, when a query needs to be evaluated.

Lifecycle

This component is initialized and destroyed at the same time the kernel is.

3.3.6 Query Processor

Functionality

The Query Processor provides access to the internal description of Triple Space data encoded using RDF data model.

The Query Processor provides query access with a query language which is based on the recommendations given in deliverable D3.2 [21] State of the art and Triple Space-specific requirements of semantic query languages. Is is responsible for decomposing a query to parts that are satisfiable by the local data store and to parts that must be forwarded to other kernels in order to fulfill the query.

Coordination Interface

The Query Processor processes queries in a synchronous request-response fashion via the space. It is responsible to process requests generated by other components.

- Event: A new query request is being inserted into the space.

  Action: The Query Processor evaluates the inserted query and selects an execution plan depending on the TS metadata. Based on the execution plan the Query Processor may need to decompose the query into subqueries. The Distribution Manager takes care of distributing the subqueries to other kernels if needed. The appropriate matching approach for queries that are satisfiable by the local data store is selected based on the TS metadata. If the matching approach selected is not supported by the Query Processor, the query is delivered to the Triple Store Adapter component.

  Source: Any component able to make queries in a language recognized by the Query Processor.

Other Interfaces

None identified so far.

Lifecycle

The Query Processor is created when the kernel is started/bootstrapped and destroyed when the kernel is stopped.

3.3.7 Metadata Manager

Functionality

The Metadata Manager is one of the core components of any triplespace implementation. As a matter of fact, the ontology-driven management is seen to be one of the major assets of semantic tuplespaces compared to traditional space frameworks. The use of ontologies provides sophisticated means for data and infrastructure processing which directly influences and improves the internal algorithms, in particular with respect to the imminent distribution and scalability issues. Understanding the meaning of information, their relationship and semantic neighborhood allows the construction of clusters and semantic overlay networks which improves the discovery efficiency and quality and thus in turn the performance of large scale implementations.

As the name already reveals, the Metadata Manager is responsible for the processing of metadata. Metadata in TripCom refers to the administrative data about the space installation, implementation and the application data (user data) published therein. This includes the modelling of spaces, data (triples and graphs) and kernels with respect to their characteristics, data access logs and distribution within the global space system.

Coordination Interface

There are two core tasks that are taken on by the Metadata Manager. First, it provides a validator service to ensure that the administrative data does not violate the model of the TS ontology. Second, the Metadata Manager serves as access provider to the metadata, i.e. it manages the creation of administrative data and guides the resolution of metadata-related queries. Both tasks are shortly discussed in more detail hereafter:

- Creation: Metadata is generally created and used by the space system only, and is thus not available to users. There are however some few dedicated methods available as part of the Triple Space API (Section 3.3.1) as well as the Management API, that allow external users to initiate the creation of metadata. `createSpace(s1,s2)` for example triggers the creation of the following triples:

```
s1 rdf:type Space .
s1 isSubspaceOf s2 .
```

  In the same way the Metadata Manager handles the requests for administrative data coming from other components of the triplespace implementation.

- Access: The metadata is stored in the same storage infrastructure as the application data, and hence also accessible in the same way. For security reasons the administrative metadata and the application data need to be kept separated and appropriate access methods have to be provided.[1] These dedicated access methods are expelled to the user in form of built-in functions that are envisioned to be integrated with the semantic templates in order to refine the queries. Assuming SPARQL-like templates a built-in function could be applied to the FILTER-statement:

```
SELECT ?s ?p ?o
FROM S
WHERE { ?s ?p ?o .
        FILTER publishedBy(P)
      }
```

  The Metadata Manager is thus responsible to interpret the `publishedBy(P)` filter and to translate it accordingly to the query format of the underlying storage infrastructure.

As already mentioned ontology-driven space management is expected to have a particularly positive effect on the processing of data in distributed settings. The Metadata Manager will thus be closely integrated with the Distribution Manager (Section 3.3.9) to model the physical distribution of data and to improve the efficiency and quality of data discovery in distributed Triple Space installations. Such tasks might require more sophisticated, in particular rule-driven reasoning that is not supported by the storage and reasoning infrastructure of Triple Space and we expect therefore that the Metadata Manager will provide its own reasoning engine, as integrate part of the component. This depends however largely on the actual tasks that are eventually expected from the Metadata Manager.

Other Interfaces

The Metadata Manager is a kernel internal component that has no need for support from external services. There are thus no additional interfaces expected, neither inbound nor outbound.

[1] The reader is referred to deliverable D2.2 for more information [18].

Lifecycle

The Metadata Manager is a core component of any Triple Space implementation and its lifecycle is thus tightly bound to the one of the kernel, i.e. startup and shutdown are aligned with the kernel startup and shutdown.

3.3.8 Transaction Manager

Functionality

The purpose of the transaction manager is to provide support for atomic transactions in a Triple Space. An atomic transaction shall, as far as it is possible, support full ACID-ity, while in the context of Triple Space some of these properties may have to be relaxed. For example, the transaction manager could support different isolation levels. Precise decisions regarding which ACID properties may be relaxed will be made in the next implementation phase of TripCom.

Furthermore, the transaction manager may support what we termed distributed transactions, which are transactions being performed by more than one client and potentially then on more than one kernel. Here, the manager acts as a proxy, taking every operation within a certain transaction and handling them atomically in the system as if there were being requested from a single client.

Coordination Interface

The component handles the following events: create transaction, cancel transaction, commit transaction and transactional operation.

Each event is registered in the integrating middleware by the TS API implementation. The manager maintains a list of active transactions. By transaction creation, it allocates the new transaction an identifier and registers it in its list. By cancel/commit, the transaction is resolved and removed from the list.

An open question is to what extent the transaction manager itself maintains local copies of data before the start of a transaction or registers of which operations take place within a transaction. This depends on the level of support for transactions in the logical implementation of the Triple Space and its physical persistent storage layer. It may be that the manager solely maintains which transactions are active and transactional operations are directly handled at the data management layers as well as rollback/committal of those operations.

The transaction manager intercepts transactional operations and "processes" them. In the case of distributed transactions, this ensures that concurrent operations in a transaction can be resolved. Importantly, the manager ensures that transactions complete correctly. This is a non-trivial requirement on the manager which will be supported by the implementation decisions to be taken in the next phase of TripCom.

Once a rollback or committal is requested, the manager ensures that all operations received in that transaction have completed and that all data management layers involved in that transaction (as it may have changed data on different nodes and in different storage layers) have successfully rolled back or committed the operations in the transaction.

Other Interfaces

If the transaction manager itself provides transactional support it may need to store local copies of data or registers of which operations were performed in a certain transaction. Otherwise, it stores only which transactions are active and which operations in those transactions are not yet completed. This storage could however be in memory or re-use the persistent storage layer of the Triple Space, hence another interface is not necessarily required.

Lifecycle

The Transaction Manager is instantiated when a create transaction request is received. It will monitor for operations taking place within that transaction and is disposed once a commit or rollback request is received, all operations within the transaction have completed, and all data management components have confirmed successful committal or rollback of changes.

3.3.9 Distribution Manager

Functionality

The data of Triple Space will be distributed physically across kernels, each of which will use some physical storage infrastructure to persistently store the data. Each kernel will be responsible for handling some of that data (including replicated data). Operations in the Triple Space will be executed on one kernel, however they may require data which is administered at another kernel. The distribution manager has the task of coordinating operations between those kernels.

Coordination Interface

A request to pass an operation onto another kernel may occur when the operation cannot be performed at the current kernel, e.g. search for knowledge which is handled at another kernel, or should be performed at another kernel, e.g. in a semantically clustered Triple Space the insertion of data which belongs at another kernel.

Generally, an operation is performed at the local kernel though in some cases the tuplespace metadata is queried first to check which kernel is relevant, e.g. if the operation applies to a certain subspace then it must be resolved which kernel handles data in that subspace. It may also be that an operation can not be answered at the local kernel. As a result, two types of request can be provided to the distribution manager: (i) a directed request, i.e. the operation is provided with the kernel it should be passed to; (ii) or an open request, i.e. the operation is provided without this indication. The distribution manager must then resolve two matters: (i) the next kernel(s) to pass the request to; (ii) monitoring the resolution of the request, either in success or failure.

This approach hides aspects of the distribution of the Triple Space from the client by employing a routing mechanism that will be used to know which kernel is the next in a chain to reach a specific kernel and to distribute a request to one or more kernels in an open request. At the most extreme case, one could flood the network with a request to every kernel, however with clustering of data the requests could be directed towards the kernels most likely to be able to resolve the request. In any case

however, there must be a restriction on the number of kernels the query is allowed to travel through (similar to a TTL known from network protocols) in order to avoid "broadcast storms" and thus network overloading.

Another issue is infinite chaining of requests. In a later implementation phase, we plan to use approaches from self-organizing systems so that e.g. a request could leave traces where it has traveled which fade over time, hence a kernel could determine if an incoming request has already been there and if enough time has elapsed to try to resolve the request again. For now, tied with a restriction of the number of kernels traveled through, each request within the system could carry a list of visited kernels in its header so that a kernel can avoid sending it back to any kernel where it has already been (such a list also helps, in the case of resolution, with returning the response back to the originating kernel).

The distribution manager is also picking up requests from other kernels. Every request from another kernel either succeeds or fails at that kernel. The distribution manager is tasked with reporting to the other kernel the result of the operation, so that a result or a failure message may be passed back to the originating kernel.

It may be that we employ some kind of existing P2P network overlay to implement this. The existing work on such networks may provide guidelines for implementation decisions such as the form of routing, in order to minimise network overload while efficiently finding the correct kernel to handle a certain operation. This idea is discussed in greater detail in Section 3.4.2.

Other Interfaces

- The distribution manager may need some store for routing information that it can efficiently inspect. Alternatively, it may make use of the Triple Space ontology defined in Work Package 2 (see [18]).
- It will have remote access to other kernels on the network.

Lifecycle

- If the kernel is part of a network, it will be started with a distribution manager.
- The manager is killed whenever the kernel is killed, sending a notification message to the neighbouring kernels that the kernel will not be available so that they can remove the kernel from their routing information.

3.3.10 Web Service Invocation

Functionality

The Web service invocation component is responsible for performing the interaction with an operation on a Web service, i.e. the execution of a service. This mainly involves the translation of Web service calls to Triple Space API operations. The translation is supported by storing graphs on the service requester's side and retrieving graphs on the provider's side.

The requesters can either be Semantic Web services or clients - requiring only minimal transformation from the internal "Semantic" data representation to RDF graphs supplied into the Triple Space API or non-Semantic Web services or clients for

which additional processing is required i.e. transformation between their (requester's and provider's) internal format and RDF (format of Triple Space) using appropriate (format) adapters.

A Web service invocation comprises the following steps which are implemented as part of the TripCom Web service binding:

- After reception of an invocation request through the *Web service API*, the security manager is asked if the requester is allowed to perform this operation.
- After successfully passing the security check, the Web service invocation component decides whether the request is from a "Semantic" or a "non-Semantic" user.
- In the latter case, proper transformation steps are applied to the request message in order to generate an RDF represenation of the request which can be transmitted over the Triple Space. When data mediation for transformed messages is required, the invocation component interacts with the mediation manager.
- After message translation, the resulting graph is stored to the space defined in the request message using the Triple Space API primitives.
- For the service provider to be notified upon insertion of a request message, an appropriate template is registered by the service provider through the TripCom binding.

The functionality mentioned above is the main part of the Web service binding for TripCom and further described in D4.1.

Coordination Interface

- Incoming events: The invocation component receives an invocation request from a TripCom user i.e. a Web service or a client through the *Web service API* as shown in Figure 3.1.
- Outgoing actions: While processing the request message, the invocation component interacts with the Triple Space API to store the request graph to the destination space. Furthermore, interaction with the security manager is required to ensure that the request complies with installed security policies. As explained above, the mediation manager may also engage in the invocation operation.

Other Interfaces

- Interfaces to mediation manager
- Interfaces to security manager

Lifecycle

As the Web service invocation component is part of the implementation of the Web service API, its lifecycle is tightly bound to it, i.e. an instance of the Web service invocation component will be created when a client implementing a Web service API initiates the Web service invocation. As soon as the invocation process is complete, this particular instance of Web service invocation will be terminated.

3.3.11 Web Service Discovery

Functionality

The Web service discovery component provides support for locating machine processable descriptions of Web services that potentially fulfill user's requirements. This mainly involves receiving the requests from its users, looking at spaces in order to find one or more Web service descriptions, and returning them to the users.

The Web service discovery component shall receive either semantically or natively encoded descriptions of the requirements from its users. In the former case, a minimal transformation from the internal "Semantic" data representation to RDF graphs is required, while in the latter case an additional level of transformation is required. The translation shall be supported by use of Adapters.

A Web service discovery comprises the following steps:

- After receiving of a discovery request through the *Web service API*, the security manager is asked if the requester is allowed to perform this operation.
- After successfully passing the security check, the Web service discovery component decides whether the request is from a "semantic" or a "non-semantic" user.
- In the latter case, proper transformation steps are applied to the request message in order to generate an RDF represenation of the request which can be transmitted over the Triple Space.
- After message translation, the resulting graph is stored for future use (i.e. reuse of goals) and also sent to the Query Processor.
- After receiving results from the Query Processor, the result is associated with the original request graph, stored for the future use (i.e. reuse of results) and returned to the requesters in their format. This may require transformation between different representation formats. During such transformation Web service discovery component interacts with the mediation manager when required.

Coordination Interface

Incoming events: Typically the incoming event for the discovery component are the request for the retrieval of Web service descriptions that potentially fulfill the user requirements. This event should be initiated through the *Web service API* as shown in Figure 3.1.

Outgoing actions: After receiving the request for the retrieval of the (Web) service descriptions, discovery component should return a set of Web service descriptions. Interaction with the security manager is required to ensure that the request complies with installed security policies. As stated above, the mediation manager may also engage in the discovery process.

Other Interfaces

- Interfaces to TripCom storage

- Interfaces to Mediation Manager
- Interfaces to Security Manager
- Interfaces to Web Service Registry
- Interfaces to Query Processor

Lifecycle

As the Web service discovery component is part of the implementation of the Web service API, its lifecycle is tightly bound to it, i.e. an instance of the Web service discovery component will be started when a client implementing the Web service API *startup*. This instance of the discovery component will be terminated as soon as the discovery process is complete and the result is returned back to the client.

3.3.12 Web Service Registry

Functionality

This component will be available as a Web service. The Web service registry component is in charge of receiving the request to publish Web service descriptions (WSDL) and Semantic Web services descriptions (WSML) and in our case transforming this description to RDF graphs. Then, the RDF graphs will be stored in the Triple Space.

Coordination Interface

- Event: publish Web service: The event is raised whenever a web service description must be published in the Triple Space. The information to publish in traditional web services is: the interface via WSDL and service contracts by free text.

  Action: The Web service description is transformed to a graph in RDF format and stored in the Triple Space.

  Source: Service API generates the event. This API is invoked by a web service client.

- Event publish Semantic Web service: The event is raised whenever a Semantic Web service description must be published in the Triple Space. The information to publish in Semantic Web services is: the interface and specifications of preconditions, postconditions, assumptions or effects via WSML.

  Action: The Semantic Web service description is transformed to a graph in RDF format and stored in the Triple Space.

  Source: Service API generates the event. This API is invoked by a Web service client.

- Event: delete Web service: The event is raised whenever a Web service description must be removed from the Triple Space.

  Action: The specified Web Service is removed from the Triple Space storage.

  Source: Service API generates the event. This API is invoked by a Web service client.

- Event: delete Semantic Web service: The event is raised whenever a semantic Web service description must be removed from the Triple Space.

  Action: The specified Semantic Web service is removed from the Triple Space storage.

  Source: Service API generates the event. This API is invoked by a Web service client.

Other Interfaces

Web service registry will not offer any external interfaces.

Lifecycle

The Web service registry component is part of the implementation of the Web service API, so its lifecycle is tightly bound to it. Hence, when the Web service API is available, the web service registry can be invoked. The Web service registry component starts whenever invoked to realize a task and terminates when its function is successfully completed (translate to RDF graphs and publishing/removing to the Triple Space through TS API).

3.4 Deployment Architecture

The deployment architecture defines the relationship between the physical world and the logical world. The physical world is composed of servers, network devices, and addressable network endpoints as well as instances of executing components. The logical world is represented by spaces and the information that is maintained in these spaces. The deployment architecture defines therefore the way how logical operations on spaces are translated into interactions between network addressable entities.

3.4.1 A Layered Deployment Architecture

Figure 3.2 represents an overview of the Triple Space deployment architecture, which is structured into four layers:

The Physical Infrastructure

The physical infrastructure is the bottom layer. It contains the physical servers, the network infrastructure and as such the network endpoints. The functionality offered by the Triple Space is embodied by software components that form the Triple Space kernel. These components are deployed on servers. We require that at least one server of the group of servers that hold the components of a kernel is an addressable endpoint on the network. The IP address of such endpoint identifies the physical infrastructure of a single kernel. The IP address is registered at a DNS.

Example: The kernel is deployed on the servers `tsc1.tripcom.it` and `tsc2.tripcom.it`.

The Component Layer

A Triple Space kernel is a single physical implementation consisting of all necessary components implementing the Triple Space API. A kernel can host multiple nodes and can itself be distributed on multiple servers. The group of servers that span a single kernel are organized in a local network site, also referred to as "cell". The cell represents a cluster of servers and is transparent to the outside world.

As described in Section 3.2 "Component Integration", the components of a single kernel are integrated via XVSM. The notion of XVSM is used in two different contexts: In the context of the "eXtensible Virtual Shared Memory" technology used to integrate the kernel components and in the context of the XVSM protocol. The XVSM protocol is used to connect the individual components of the kernel with the XVSM system. The XVSM system is embodied by an executable service that forms the centerpiece of the XVSM middleware. The XVSM system implements the shared data container described in 3.2. The XVSM protocol provides data exchange and coordination capabilities between the components and the XVSM system. A single Triple Space kernel integrates all its constituent components using a single XVSM system, which is accessible through an XVSM protocol endpoint. This means that a kernel is exposed through a unique XVSM protocol endpoint. We are therefore able to identify a kernel by a unique, XVSM-based URL.

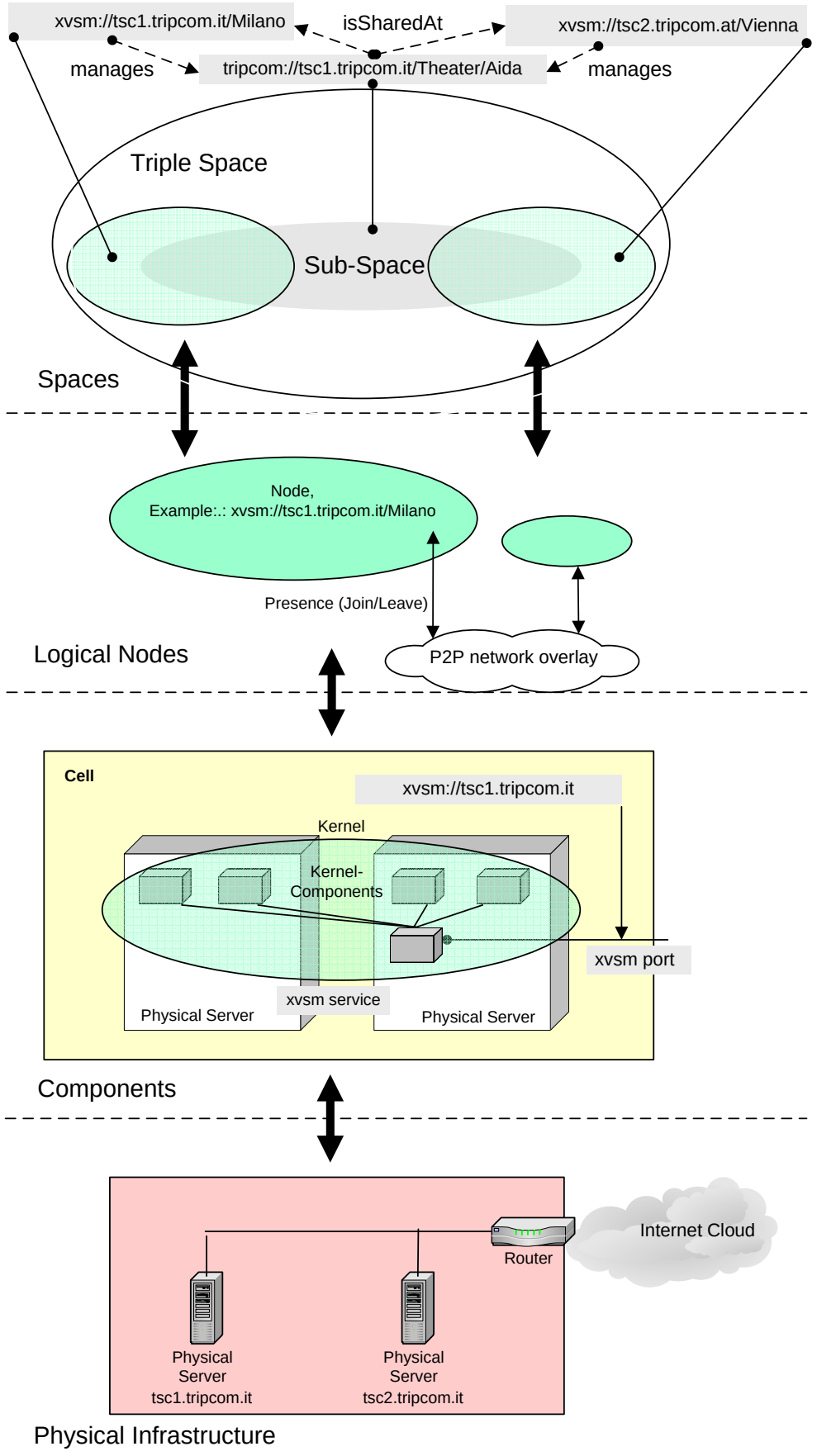

Figure 3.2: Triple Space Architecture – Deployment Overview

Example: The URL `xvsm://tcs1.tripcom.it:5000` represents the kernel deployed on `tsc1.tripcom.it` and `tsc2.tripcom.it` with an XVSM protocol endpoint listening on port 5000. Although standard port assignments for the XVSM protocol have not been defined yet, we can assume such a standard assignment and simplify the URL to the form `xvsm://tcs1.tripcom.it`.

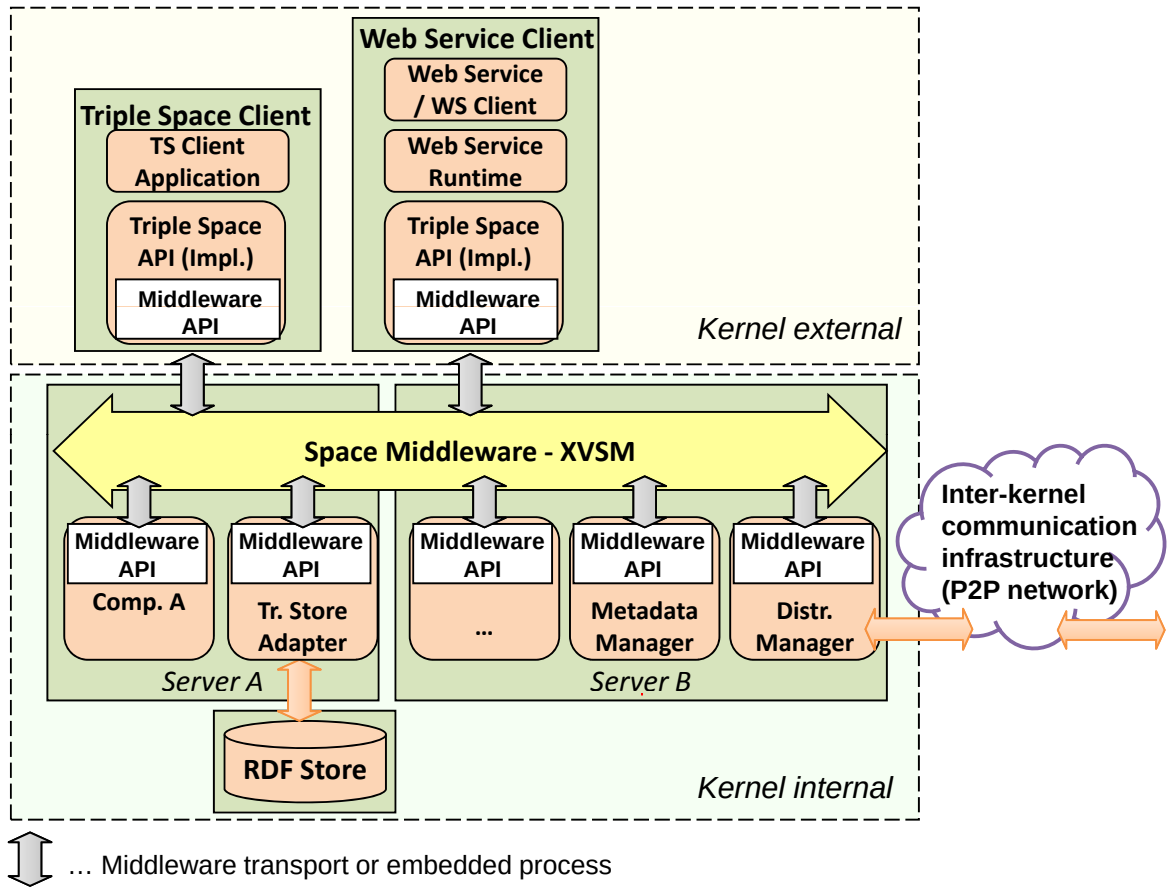

Figure 3.3: Triple Space Architecture – Deployment View

Figure 3.3 depicts the deployment of a kernel. The Triple Space client application uses the TS API to access the Triple Space, either directly via the XVSM protocol or through additional translation via Web services as described in deliverable D4.1 [23].

The TS API requests are translated into tuple-like data structures and written into the internal XVSM space. The various components interacting with the XVSM space read these data structures and deliver their responses back into the XVSM space.

While some components connect to the XVSM system remotely, others may reside on the same physical server or even share the process space with the XVSM system itself. All components, with exception of the Distribution Manager, are connected to exactly one XVSM system. The Distribution Manager connects multiple kernels and is thus connected to multiple XVSM systems. The Distribution Manager is responsible for message routing and acts according to semantics defined in the Triple Space metadata, based on the TS ontology (see deliverable D2.2 [18]). All physical servers that contain components of a single kernel must be deployed within the security perimeter of this kernel. The precise defintion and implementation of the security perimeter will be provided by Work Package 5.

The Logical Nodes Layer

While the component layer exposes a physical kernel, the nodes layer exposes a logical node that supports the same TS API as the kernel. One kernel can host an arbitrary number of nodes. A node is managed under a single authority and is identified by a unique logical identifier. In compliance with the URI syntax, this identifier consists of a scheme (xvsm), an authority (the DNS name of a kernel) and a path that identifies a node relative to its corresponding kernel.

Example: The full name for a node "Milano" on kernel `tcs1.tripcom.it` would be `xvsm://tcs1.tripcom.it/Milano`.

The concept of a logical node is analogous to the concept of virtual hosts in contrast to a physical Web server. The concept of the logical node allows to separate between deployment on physical systems and the logical structure as described by the TS metadata. Thus, a logical entity described by the TS metadata, like a sub-space, can be redeployed on various physical infrastructures.

The introduction of such concept introduces two capabilities:

- **Management:** A single physical kernel might be partitioned into multiple nodes - similar to a Web server being partitioned into individually managed virtual sites. Each node can be managed individually.
- **Logical addressing:** The address of a logical node is independent from the physical deployment. A logical node is identified by a URI. However, the name of a logical node must be unique among all nodes in the Triple Space. We need to guarantee the uniqueness without recurring to a central registry. Therefore, we propose to use a URI identifying a logical node being an extension of the managing kernel's URL.

The TS ontology (see deliverable D2.2 [18]) that provides the means to describe the relationship between spaces and kernels does not differ between kernel and logical node since the ontology deals with the logical world. The differentiation is required on the level of the deployment in a distributed configuration.

The Space Layer

Triple Space is conceived as multiple non-overlapping spaces, which are structured into hierarchical trees of spaces as described in the TS ontology. A space may span multiple nodes — and as such kernels — independent of the physical location of the nodes, meaning any node defined on any kernel may be part of a space. The Triple Space ontology described in D2.2 [18] provides the mean to describe the space, its relationship to other spaces and its mappings to nodes. The metadata manager provides the capability to inquire on these descriptions.

A sub-space is identified by a name, which must be space wide unique. Triple Space uses URIs to identify spaces. In order to guarantee the uniqueness of the URI, it must be derived from the URI of a contributing node, for example from the URI of the node that is the initial creator of the sub-space. Such mechanism is described in Deliverable D3.1, Chapter 6 "Requirements for a Triple Space API" [20]. Since a

space is not directly addressable but rather accessed via the TS API, the URI prefix is changed to "tripcom".

Example: A subspace is identified by: `tripcom://tsc1.tripcom.it/Milano`

3.4.2 Moving Toward a Distributed, Scalable Deployment Architecture

The architecture presented until now was mainly concerned with the architecture of a single kernel. In order to support the distribution of a space over multiple kernels the deployment architecture requires at least a distributed directory capability allowing Distribution Managers in a large number of kernels to find each other and to exchange information. The deployment architecture proposes the introduction of a peer-to-peer (P2P) [4] network overlay technology allowing multiple kernels to cooperate.

The role of the network overlay is evident by arranging the necessary functions for the distributed architecture into three functional groups:

Network Overlay: The task of the network overlay is to maintain a distributed directory of node/kernel-to-space mappings and to allow individual nodes to find each other. The Triple Space ontology described in D2.2 provides the base for such mapping by defining the relationship between spaces and kernels. The network overlay allows to retrieve and to address each kernel that is part of a space and each space that is maintained in a kernel. We assume that the network overlay provides an advertisment service that allows to publish the mappings.

Distribution Manager: The Distribution Manager component has the responsibility to connect to the network overlay, to perform join and leave operations against it and to publish it's node/kernel-to-space mappings into an advertisement service. Usually a TS API request is specified with a space as argument (i.e. a particular space or the global space), allowing to forward the requests to the appropriate kernels. However it will be necessary to define forwarding policies in case that no space or the global Triple Space is specified. These policies should limit the forwarding of a request to a reasonable amount of nodes, avoiding the flooding of the network with forwarded requests.

Metadata Manager: The Metadata Manager creates space descriptions according to the Triple Space ontology and allows to inquire on them.

The network overlay provides a distributed directory function allowing the federation of kernels. There are three basic possibilities to implement the distributed directory:

- In form of a centralized directory service. This solution is neither scalable nor fault tolerant and requires the establishment of a central authority
- In form of a hierarchical tree of directory servers similar to a tree of DNS servers or X.500 Directory Service Agents. A hierarchical tree of servers would require not only a strictly hierachical namespace but also a strict organizational hierarchy of authorities, with a central authority at the top. Although such hierarchies seem to contradict the idea of an open, Web-scale platform, the use cases

selected in WP8a and WP8b would not be hampered by a formal hierarchy. On the contrary, the security systems required for the protection of medical data and of copyrighted content would probably benefit from a very controlled and hierarchically organized environment.

- In the form of a structured P2P network overlay supporting the publication of objects based for example on a distributed hash table technique.

We favor the P2P technology for the implementation of the network overlay. The reasons to build the Triple Space on top of a P2P network overlay are:

- A P2P network overlay does not require any central coordinating or controlling entity. Such central entity would contradict the idea of a space that is co-owned by its creator and by the nodes that joined into the space.
- The P2P network overlay does not force any predefined structure on the network, allowing nodes to be added at any point in the network overlay. A node can add itself to the network overlay as soon as it knows any single node that allows the node to join.
- A P2P network overlay may provide resiliency against failures of individual nodes, as long as some remaining node supporting a specific space is able to respond to requests addressing this space.
- The growth of a P2P network is not limited by the capacity of an individual node.

However, the introduction of a P2P concept is insufficient to define the required distributed mechanisms. Open issues not yet addressed in the current architecture description arise from examining the lifecycle of a space:

- Space creation: A space is created by a specific node using the "create" primitive. The local Metadata Manager creates the description of the new space. The "create" primitive takes a parent space as an argument, indicating that a space is created as a subdivision of a larger space. We can therefore assume that spaces form a rooted tree, with the "Triple Space" as root. The network overlay will maintain the node/kernel-to-space mapping into its distributed directory, by using for example a distributed hash table (DHT). Thus, the resolution of a TS-API operation is straight forward as long as the operation addresses explicitly a specific space. The problem arises when a TS API operation addresses arbitrary spaces, for example: "Read all triples authored by X?". Neither the necessary strategies for the efficient execution of operations involving arbitrary spaces nor the role of the Metadata Managers in this case have been investigated at this point.
- Joining/leaving: Until now, we assumed a space to be created and maintained in a single node. Under such an assumption, scalability is achieved by "scaling-up", i.e. by adding more processing power to the kernel supporting the node. In order to reach "Web-scale" qualities in terms of scalability and fault-tolerance, we must introduce a "scaling-out" strategy, where multiple nodes in cooperating kernels share a single space. This means that a node may join to an existing

space, allowing to process TS API operations in parallel by multiple nodes. The cooperation of nodes would entail the full or partial replication of data between nodes. Such implementation strategies have been succesfully used in many distributed lookup systems as illustrated for example in X.500 by the shadowing of X.500 Directory Service Agents through the "Directory Information Shadowing Protocol"(X.525). Replication strategies are also the key feature used by Web-scale file sharing networks like BitTorrent, where every user is automatically turned into a replication server.

The introduction of such mechanisms require the definition of policies governing the cooperation of multiple nodes supporting a single space. These policies have not been included in the Triple Space ontology and these issues remain to be investigated.

- Space destruction: The TS API provides a space destroy primitive. Once some cooperation strategy between nodes is implemented, policy definitions are required to govern the destroy operation. Open issues are: which of the participating nodes of a space is able to execute a destroy operation and what happens to the remaining nodes.

The open issues are scheduled to be addressed in the next phase of the project which focuses on distribution and scalability of Triple Space. The currently proposed architecture limits itself by not precluding any solution of these open issues. The choice of a fitting P2P infrastructure is also scheduled for the upcoming project phase. However, some characteristics of a P2P infrastructure can be defined upfront:

Support for a Distributed Directory: The P2P network overlay must support the publication of objects as well as the resolution and routing to published objects. A good example for such a capability is found in Tapestry [27].

Presence Service: The Distribution Manager forwards requests to the appropriate kernel. The presence service allows to route the requests to live kernels and to avoid delays. Such services are usually built as applications on top of basic P2P network overlays.

Security Hooks: Cooperating kernels must be able to trust each other. The trust association mechanisms are implemented by the Security Manager. The communication channels offered by the P2P network overlay must support adequate transport security like SSL/TLS. The security mechanisms used by the P2P system proper, must be plugable in order to delegate security requests to the Security Manager.

3.4.3 Name Resolution Walk Through Example

The following example illustrates the use of sub-spaces and the associated name resolution process (we assume a situation where all participants have joined into a common space, i.e. we do not address the problem of advertising and joining of a space):

Let's assume the world of theaters and of theater tickets identified by `tripcom://tsc1.tripcom.it/tickets`. Within this world we identify the sub-space of "Aida"

performances. Aida is performed in Milan and in Vienna as a shared co-production. The Milan performances are given in "LaScala", while the Vienna performances are given in the "Staatsoper" (Vienna State Opera).

The Milan and Vienna organizations are considered as two independent authorities: these organisations maintain two separated kernels. The logical nodes maintained on these kernels are addressable in the P2P overlay in the form: `xvsm://tsc1.tripcom.it/tickets/Aida/Milano` and `xvsm://tsc99.at/tickets/Aida/Vienna` respectively. The sub-space representing the tickets for the common production of "Aida" is identified by `tripcom://tsc1.tripcom.it/tickets/Aida`.

A client requiring access to the sub-space may contact any kernel via the TS API. The arguments of the TS API requests contain a (sub-) space URI as described in the TS API definition. The kernel decides if the request can be processed locally by reasoning on its own TS metadata. The result of the reasoning process performed by the Metadata Manager may result in the invocation of the Distribution Manager. The Distribution Manager would then forward the request to a kernel that is able to process the request.

The Distribution Manager computes the list of nodes participating in a sub-space `tripcom://tsc1.tripcom.it/tickets/Aida` using information replicated in the P2P infrastructure. The Distribution Manager uses P2P advertisements to route the request to the kernels `xvsm://tsc1.tripcom.it/tickets/Aida/Milano` and `xvsm://tsc99.tripcom.at/tickets/Aida/Vienna`. The P2P infrastructure has the capability to route to the kernels represented by the respective URIs (`xvsm://tsc1.tripcom.org.it` and `xvsm://tsc99.tripcom.at`).

4 Conclusion

The Triple Space reference architecture sketched in this paper describes the roles and integration of the necessary components to achieve the goals of Triple Space. The architecture proposes an integration on two levels:

First, integration of a single Triple Space kernel based on an eXtensible Virtual Shared Memory (XVSM) technique. The use of the XVSM technology provides us with a flexible and resilient integration infrastructure. Using space based coordination principles, each individual component is able to contribute autonomously to the fullfilment of a TS API request. The implementation of the Triple Space benefits therefore from the same decoupling principles that Triple Space offers to its clients. This leads to a flexible architecture, allowing the addition, replacement, and retraction of components even at runtime.

Second, integration of multiple Triple Space kernels into a large network using a peer-to-peer network overlay. The details of this integration will be elaborated in the next phase of the project. The proposed architecture provides the basic elements for the implementation of a distributed infrastructure without the necessity of any central control.

The main requirements like scalability, reliability, interoperability, and security are the major driving forces for the architecture. However, the architecture reflects also the phased implementation of the Triple Space. The architecture was designed to add features incrementally moving gradually from a simple prototype implementation to a more advanced implementation.

We expect that the combination of both integration techniques in conjunction with the defined components provides the foundation for a Web-scalable Triple Space infrastructure.

References

[1] Dario Cerizza, Emanuele Della Valle, doug foxvog, David de Francisco, Reto Krummenacher, Henar Munoz, Martin Murth, and Elena Paslaru-Bontas Simperl. State of the art and requirements analysis for sharing health data in the ts. TripCom Deliverable D8B.1, 2007.

[2] TripCom Consortium. Triple space communication, Annex I. – Description of Work. FP6 - 027324, November 2005.

[3] David de Francisco Marcos, Daniel Wutke, Daniel Martin, Andreas Harth, and Martin Murth. Requirements analysis and architecture profile for eai applications. TripCom Deliverable D8A.1, 2007.

[4] B. Traversat et al. The project jxta virtual network. available from: `http://www.jxta.org/docs/JXTAprotocols.pdf`, May 2001.

[5] Tony Andrews et al. Business Process Execution Language for Web Services, version 1.1, July 2002. Available at: http://www.ibm.com/developerworks/library/ws-bpel.

[6] Vassil Momtchev et al. Specification of the store architecture and interfaces. TripCom Deliverable D1.2, 2006.

[7] eva Kühn. Fault-tolerance for communicating multidatabase transactions. In *Proceedings of the 27th Hawaii International Conference on System Sciences (HICSS)*, volume 4, Wailea, Maui, Hawaii, 1994.

[8] eva Kühn. The zero-delay data warehouse: Mobilizing heterogeneous databases. In *Procs. of the Very Large Databases (VLDB Conference), ACM, IEEE*, November 2003.

[9] eva Kühn, Johannes Riemer, and Gerson Joskowicz. eXtensible virtual shared memory (XVSM) – architecture and application. Technical report, Vienna University of Technology, Institute of Computer Languages, E-185-1, June 2005.

[10] Dieter Fensel, Reto Krummenacher, Omair Shafiq, Eva Kuehn, Johannes Riemer, Ying Ding, and Bernd Draxler. TSC - Triple Space Computing. *e&i Elektrotechnik und Informationstechnik (forthcoming)*, 124(1/2), Febuary 2007.

[11] Eric Freeman, Ken Arnold, and Susanne Hupfer. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1999.

[12] David Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[13] David Gelernter and Nicholas Carriero. Coordination languages and their significance. *Commun. ACM*, 35(2):97–107, 1992.

[14] GigaSpaces. Gigaspaces data grid - caching. Technical report, GigaSpaces - White Paper, available from: `http://www.gigaspaces.com/os_papers.html`, 2005.

[15] GigaSpaces. Gigaspaces enterprise application grid version 4.1 documentation. available from `http://www.gigaspaces.com`, 2006.

[16] Gregor Hophe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions*. Addison-Wesley, 2004.

[17] Gerson Joskowicz, eva Kühn, and Martin Murth. The XD model: Extending XML and DOM to standards based coordination. In *Proceedings of the 10th IASTED International Conference on Software Engineering and Applications (SEA), Nov. 13-15, Dallas, USA*, pages 146–152, 2006.

[18] Reto Krummenacher, Elena Simperl, Vassil Momtchev, Lyndon J.B. Nixon, and Omair Shafiq. Specification of triple space ontology. TripCom Deliverable D2.2, March 2007.

[19] A. Lederer. The database replication coordination design pattern. Master's thesis, Institute of Computer Languages, Vienna University of Technology, 2004.

[20] Lyndon Nixon, Elena Paslaru Bontas Simperl, Reto Krummenacher, Francisco Martin-Recuerdaa, Martin Murth, Geri Joskowicz, and eva Kuhn. Specification and implementation of a semantic linda model. TripCom Deliverable D3.1, 2007.

[21] Janne Saarela, Tommi Koivula, Lyndon Nixon, and Axel Polleres. State of the art and triplespace-specific requirements of semantic query languages. TripCom Deliverable D3.2, 2007.

[22] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[23] Brahmananda Sapkota, Dario Cerizza, Emanuele Della Valle, Daniel Martin, Martin Murth, Omair Shafiq, Henar Munoz, and Andrea Turati. Architectural integration of triple spaces with web service infrastructures. TripCom Deliverable D4.1, 2007.

[24] F. Scharffe and J. de Bruijn. A Language to Specify Mappings Between Ontologies. *Proc. of the Internet Based Systems IEEE Conference (SITIS05)*, 2005.

[25] Nati Shalom. The scalability revolution: From dead end to open road. Technical report, GigaSpaces - SBA Concept Paper, available from: `http://www.gigaspaces.com/os_papers.html`, 2007.

[26] Sun. Sun microsystems: JavaSpaces service specification. Jini Network Technology Specifications (available at `http://www.sun.com/software/jini/specs/`), 2003.

[27] The chimera and tapestry home page. `http://p2p.cs.ucsb.edu/chimera/html/home.html`, 2006.

[28] Robert Tolksdorf and Dirk Glaubitz. XMLSpaces for coordination in web-based systems. In *Procs. of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure For Collaborative Enterprises, IEEE Computer Society*, 2001.

A Implementation Plan

A.1 Implementation Plan

Each component of the Triple Space is supervised by one TripCom partner. This partner is responsible for managing the implementation work and delivering the component implementation and documentation in time. The partner also acts as a contact for any issues related to this component (see Table A.1).

Component	Supervision	Phase 1	Phase 2	Phase 3
System Integration	WP6/TUW	-	T6.5 (19-24)	T6.7 (29-33)
Triple Space API	WP3/FUB	T3.2 (6-12)	-	T3.7 (29-33)
Triple Store Adapter	WP1/ONTO	T1.3 (7-12)	T1.6 (13-18)	-
Triple Space Manager	WP2/FUB	T2.3 (7-12)	T2.6 (19-23)	T2.8 (24-28)
Security Manager	WP5/CEFRIEL	-	T5.3 (19-24)	T5.6 (29-36)
Mediation Manager	WP4/TID	-	T4.5 (19-24)	T4.6 (28-36)
Metadata Manager	WP2/LFUI	-	T2.6 (19-23)	-
Query Processor	WP3/PROFIUM	-	T3.5 (19-23)	T3.6 (24-28)
				T3.7 (29-33)
Transaction Manager	WP3/FUB	-	T3.5 (19-23)	T3.7 (29-33)
Distribution Manager	WP2/FUB,ONTO	-	T1.7 (19-24)	T2.8 (24-28)
		-	T2.6 (19-23)	
Web Service Invocation	WP4/NUIG	-	T4.2 (13-24)	T4.6 (28-36)
Web Service Discovery	WP4/NUIG	-	T4.2 (13-24)	T4.6 (28-36)
Web Service Registry	WP4/TID	-	T4.3 (13-18)	-
Use Case 8A	WP8A/TID	-	T8A.5 (13-22)	T8A.9 (31-36)
Use Case 8B	WP8B/CEFRIEL	-	T8B.3 (16-22)	T8B.7 (31-36)

Table A.1: Implementation Plan

TripCom's overall implementation will be carried out in three different phases:

Phase 1 – Prototypical implementation: The first prototypical implementation serves as a proof-of-concept for the Triple Space API semantics. It therefore only implements the most basic set of components (namely the Triple Space API, Triple Store Adapter and core management functions). The planned timeframe for Phase 1 is M6 to M12.

Phase 2 – Implementation of core functionality: Phase 2 leads to a full prototype and therefore implements all components defined in the Triple Space architecture. This includes a distributed Triple Space infrastructure, specification of WSML and Rule representations in the Triple Space, implementation of a semantic matching component, prototypes of security and trust components, the specification of component interfaces, revision of the Triple Space API and prototypical implementations of both use case scenarios (cf. [2]). Phase 2 is scheduled to last from M13 to M24, thus allowing a duration of 12 month in total. During the work on Phase 2, thorough tests and analysis tasks will be carried out and serve as input for the final implementation phase.

Phase 3 – Implementation of advanced TS features: The results of the second phase drive this prototype refinement phase. Moreover, some advanced functionality of Triple Space will be implemented. This includes the implementation

of a distributed semantic query tool, refinement of Triple Space architecture, integration of the Triple Space environment with WSMX, the development of a Triple Space demonstrator application, implementation of security and trust model, and an in-deep evaluation of the entire Triple Space infrastructure, its components, and the use case implementations (cf. [2]). The planned timeframe for Phase 3 is M24 to M36, allowing 13 month of time to implement all tasks.

A.2 Triple Space Implementation Tasks

Table A.2 shows a detailed list of implementation tasks and the corresponding implementation phases mentioned before.

Task	Duration	Description
Prototypical implementation (year 1)		
T1.3	7-12	First implementation: Basic storage infrastructure based on RDF stores
T2.3	7-12	Initial implementation of RDF triple semantics in tuples
T3.2	6-12	Implementation of a semantic Linda model
Implementation of core functionality (year 2)		
T1.6	13-18	Second Implementation: Abstraction for bindings of non-triple-based data stores
T1.7	19-24	Third implementation: Linking of Triple Spaces; distributed storage
T2.6	19-23	Implementation of a semantic clustering solution for scalability of Triple Spaces
T3.5	19-23	Implementation of semantic matching in a distributed (semantically clustered) Triple Space
T4.3	13-18	Implementation of Web service registry mechanisms in a Triple Space
T5.3	19-24	Implementation: Early prototype of security and trust components
T6.5	19-24	First implementation: Prototypes for all components and basic component integration
Implementation of advanced TS features (year 3)		
T2.8	24-28	Implementation of a self-organization solution for scalability of Triple Spaces
T3.6	24-28	Implementation of a distributed semantic query tool
T3.7	29-33	Evaluation and refinement of implementation
T4.6	28-36	Integration and evaluation of Triple Space within WSMX (results in prototype)
T5.6	29-36	Second Implementation: Refinement and final prototype
T6.7	29-33	Second implementation: demonstrator and use case integration

Table A.2: Triple Space implementation tasks

A.3 Use Case Implementation Tasks

The TripCom infrastructure will be employed in two different use cases. One centered around the case of the "European Patient Summary", the other one is a use case in the domain of Enterprise Application Integration (EAI), dealing with Digital Asset Management (DAM). Both are described in the respective TripCom deliverables D8A.1 [3] (EAI) and D8B.1 [1].

Task	Duration	Description
Implementation of core functionality (year 2)		
T8A.5	13-22	Prototype development of a TripCom application towards the EAI scenario
T8B.3	16-22	Prototype of a TripCom application towards the eHealth scenario
Implementation of advanced TS features (year 3)		
T8A.9	31-36	Prototype implementation refinement
T8B.7	31-36	Prototype implementation refinement

Table A.3: Use Case implementation tasks