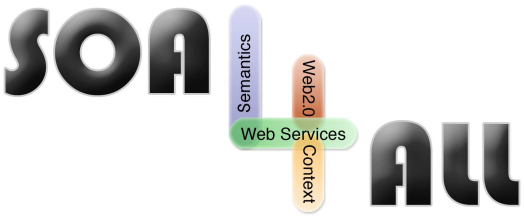

Project Number: **215219**

Project Acronym: **SOA4All**

Project Title: **Service Oriented Architectures for All**

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D1.3.2A Distributed Semantic Spaces: A Scalable Approach To Coordination

Activity N:	Activity 1 - Fundamentals and Integration Activity	
Work Package:	WP1 - SOA4All Runtime	
Due Date:		M12, resubmitted M18
Submission Date:		14/09/2009
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		UIBK
Revision:		1.1
Author(s):	Reto Krummenacher UIBK Imen Filali INRIA Fabrice Huet INRIA Francoise Baude INRIA	

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	2008-11-11	Initial TOC	Reto Krummenacher (UIBK)
0.2	2008-12-16	Introduction, content to Sections 2.3 and 3.1	Imen Filali (INRIA), Reto Krummenacher (UIBK)
0.3	2008-02-25	Final release for internal review	All
0.4	2008-02-27	Integration of feedback by Jacek Kopecky (UIBK)	All
0.5	2008-03-03	Integration of review by Dong Liu (OU)	All
1.0	2008-03-06	Final release (submission)	Reto Krummenacher (UIBK)
1.1	2008-08-15	Quality improvements (resubmission)	Reto Krummenacher (UIBK), Fabrice Huet (INRIA)
1.1	2009-09-14	Final editing	Malena Donato (ATOS)

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms

Acronym	Definition
API	Application Programming Interface
BC	Binding Component
BPEL	Business Process Execution Language
CAN	Content Addressable Network
D	Deliverable
DSB	Distributed Service Bus
EC	European Commission
EJB	Enterprise Java Beans
ESB	Enterprise Service Bus
FP	Framework Program
FP7	The 7th Framework Program
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineerings
M	Median, Milestone
OWL	Web Ontology Language
P2P	Peer-to-Peer
RDF	Resource Description Framework
SE	Service Engine
SOA	Service-Oriented Architecture
SOA4All	Service-Oriented Architectures for All
SOAP	Simple Object Access Protocol
T	Task
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WP	Work Package
WSMX	Web Service Execution Environment

Executive Summary

This second deliverable on Semantic Spaces in SOA4All presents the project's approach and solutions to a distributed and scalable semantics-driven data management infrastructure. Scalability is achieved by distribution of semantic data across machines and by integration of data sources via federations and space hierarchies. The Semantic Space infrastructure manages independent and distributed repositories by means of state-of-the-art peer-to-peer overlay technology. A Chord index ring is put in place to manage the access to distributed virtual Semantic Spaces, while the RDF data of each individual space is indexed by means of a three-dimensional CAN network (one dimension per resource of an RDF triple). This procedure ensure O(log N)-discovery of spaces, and the same logarithmic scalability in terms of data retrieval from a space.

While spaces are fixed data containers that are put in place by administrators, and that are identified by URIs and indexed in the P2P-overlay, federations are temporary scopes for the management of RDF artefacts that are either created on the fly by applications and space users, or that are implicitly enabled by means of structural metadata. Structural metadata gives meaning to relationships between spaces. In that way, a request is not only resolved against a given space, but also against all known spaces with a predefined connection (determined by administrators, or eventually derivable from the data that is contained in the spaces).

The Semantic Space infrastructure of SOA4All exposes a simple set of operations for the publication of RDF data, the querying of data via SPARQL or triple patterns, the removal of data, and the subscription to publication-events in form of queries or triple patterns. Furthermore, a set of management operations are put in place for the creation and deletion of spaces and federations.

Although, the Semantic Space exposes only one API, applications are enabled to target specific shares of the overall information space by means of the aforementioned virtual spaces. In that way, no matter what space is addressed, no matter what non-functional properties this space has, the access patterns remain the same. Moreover, the concept of information spaces inherits an important linear scalability notion, as the application layer is fully decoupled from the data management layer via this Semantic Space API. Adding more processing units linearly adds more processing power. This is a characteristic that is very important for ensuring scalability in SOA4All, and is observed in any space-based infrastructure [12]. This fact is an important motivator for the incorporation of the Semantic Space infrastructure into the Distributed Service Bus (cf. Deliverable D1.4.1A). A further noteworthy aspect of this approach to semantic data management is the inherited virtualization: no matter how the data is indexed and stored, no matter were it physically resides, the access methods remain the same and only depend on the very simple Semantic Space API.

The deliverable presents not only the conceptual specification of the SOA4All Semantic Space infrastructure, but positions it clearly within the project and within the scope of related efforts and space-based platforms for the Semantic Web. Furthermore, the deliverable presents implementation guidelines that ensure successful realization within the next six month of SOA4All.

1. Introduction

Originally, Semantic Spaces were intended as a means to decouple distributed Semantic Web applications in terms of time, space and reference. In the case of Triple Space Computing, the goal was moreover to offer (Semantic) Web services a publication platform that exposes World Wide Web and Web2.0-like interaction patterns to machines; i.e., that enables a “persistent publish and read” paradigm for the collaboration of services. In principle, Semantic Spaces is an extension to space-based computing with semantics. However, as traditional tuplespaces are mostly local platforms used to coordinate processes or a small set of machines, Semantic Spaces must take into account further non-space-based computing typical challenges like scalability, distribution or openness.

The primary benefits of Semantic Spaces are twofold. First, it allows semantic applications to coordinate over a blackboard-style infrastructure without leaving the semantic layer, and secondly it exploits semantic technology to improve the matching and querying behaviour of space-based computing systems by means of inference, reasoning and semantic matchmaking. In order to profit from the semantics of the published data, all stored information must be represented in RDF, and consequently the interaction patterns are based on the exchange and sharing of RDF triples in Semantic Spaces.

In SOA4All, Semantic Spaces are used for tow main purposes: i) for storing semantic artefacts, and ii) for asynchronous, publication-based communication between Distributed Service Bus nodes or platform services (cf. Deliverable D1.4.1A). In the case of storage, the Semantic Space infrastructure is primarily used as distributed repository for semantic description of services (SOA4All platform services, but also published business services) and processes, and as shared memory for RDF-based contextual data and monitoring data that is gathered at the level of the service bus.

In the context of the Semantic Space task (WP1, T1.3), this deliverable presents the SOA4All Semantic Space infrastructure, as it will be implemented as part of the SOA4All Distributed Service Bus. The Semantic Space infrastructure provides a blackboard or shared memory-based storage, communication and coordination platform for SOA4All platform services that rely on the exchange and persistency of RDF-based semantic data. Moreover, the functionality of the Semantic Space is also offered to users and business services via respective Binding Components that are exposed by the Distributed Service Bus. Detailed specifications on the integration of the Semantic Space infrastructure and the Distributed Service Bus are, as indicated above, given in Deliverable D1.4.1A.

1.1 Purpose and Scope

The goal of this deliverable is to the specification of the SOA4All Semantic Space infrastructure in terms of architecture, and implementation guidelines. The implementation of a first release of the infrastructure is subject to the period M13 – M18 and will be delivered in form of a prototype with Deliverable D1.3.1B. Consequently, this deliverable has two main parts with each its distinct purpose and scope. First, we present the general architecture of the Semantic Space with the exposed operations and functionality, and a detailed specification of the internal components and layers. The second part provides more technical details and explains how the Semantic Space infrastructure will be realized by help of peer-to-peer technology. Note however, SOA4All is not doing any peer-to-peer research in scope of T1.3, but merely exploits successful results from the peer-to-peer community in order to realize a scalable and highly distributed Semantic Space infrastructure. Moreover, the technology and implementation specification concentrates on the Semantic Space as such; the integration of the infrastructure with the SOA4All Distributed Service Bus, and hence its use within the overall SOA4All architecture and runtime is specified in Deliverable D1.4.1A.

1.2 Structure of the document

After this first introductory section, that provides a general overview of the deliverable, its purpose, scope and structure, Section 2 describes the Semantic Space architecture. First, we provide a comparison and contrast discussion with past Semantic Space projects, in particular the EC FP6 STREP TripCom that worked towards a global Triple Space implementation (cf. http://www.tripcom.org). This part describes the synergies, benefits, but certainly also the differences between TripCom and the SOA4All Semantic Space infrastructure. Furthermore, it depicts important drawbacks and problems with the solution of TripCom compared to the needs of SOA4All. These issues lead to the SOA4All-specific Semantic Space architecture that is described in Section 2.2; an architecture that takes in particular the eminent scalability, openness and distribution challenges of service computing at Web scale into account. Section 3, as stated earlier, provides technology and implementation details for the realization of the global Semantic Space infrastructure. First, Section 3.1 explains how we achieve scalability through distribution by means of peer-to-peer technology, while Section 3.2 elaborates on the idea of federated spaces, as a means for data integration and coordination across multiple domains; diverse domains in terms of spaces and physical origin of resources. Section 4 extends the presentation of the more technical aspects by presenting our approach towards event-driven communication over Semantic Spaces. Finally, Section 5 concludes this second deliverable on Semantic Spaces.

1.3 Methodology

Based on the requirements and application scenarios that were derived in course of the first six months of SOA4All, the work of this deliverable aims for a Semantic Space infrastructure that copes with the needs and uses of service computing at Web scale and the various platform services of the SOA4All runtime. The requirements and application scenarios were specified in Deliverable D1.3.1 that was released in M6 of the project. As already explained in this first deliverable of task T1.3, the Semantic Space task can profit from the work done in course of the EC FP6 TripCom project. The synergies were already partly discussed in D1.3.1, and are again, and more specifically, evaluated in Section 2.1 of this deliverable. TripCom established many highly valuable specifications, but more importantly, SOA4All can profit from the lessons learned and technological insights that were gained during the runtime of TripCom. This input is considered in the process of turning the requirements and applications scenarios into a SOA4All Semantic Space implementation. In particular, SOA4All must pay further tribute to scalability and distribution issue of service-oriented architectures at Web scale. For this reason, there is a strong focus on exploiting peer-to-peer technology.

2. Architecture Overview

The idea of applying space-based technology to the sharing of Semantic Web data, or the application of blackboard-style coordination for collaborative problem solving in artificial intelligence are not new concepts. In particular, the former was investigated in a recently terminated research project called TripCom (http://www.tripcom.org). The Semantic Space infrastructure of SOA4All that is subject to task T1.3 of WP1 builds on conceptual and technological insights and lessons learned from that project. In order to make the synergies and differences well understood, and in particular in order to emphasize on the added value of the work done in SOA4All, the next section (Section 2.1) provides a comparison and contrast discussion.

In SOA4All, Semantic Spaces are applied to ease the integration and management of distributed RDF artefacts across the whole SOA4All platform. The spaces provide the primary communication and collaboration service for various Platform Services and Business Services that are running within and on top of the SOA4All platform. The Service Repository uses Semantic Spaces for the realization of scalable storage infrastructure, the monitoring facilities of the SOA4All Distributed Service Bus (DSB) publishes events and facts to the space infrastructure to ensure global access to contextual information and to enable decentralized management, processing and use of the context data about the bus and services. Besides these core application scenarios, Semantic Spaces deliver a flexible, scalable and simple to use core service for the realization of various other SOA4All aspect that rely on sharing, exchanging and storing RDF data.

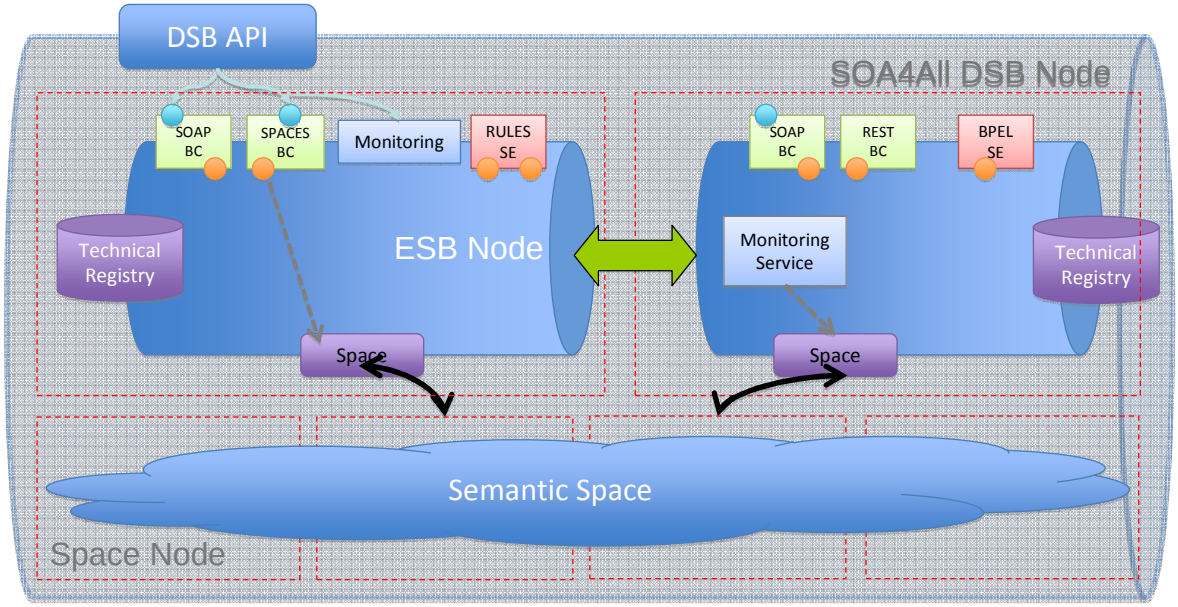

Figure 1: Semantic Space Infrastructure within the SOA4All Distributed Service Bus

The Semantic Space infrastructure that is presented in this deliverable will be realized as an integrated part of the SOA4All Distributed Service Bus, and enhances existing state-of-the-art enterprise service bus (ESB, [6]) solutions (in the case of SOA4All the PEtALS ESB; http://petals.ow2.org) with means for event-driven Web-style communication, coordination and data sharing. As Figure 1 depicts, the Semantic Spaces are distributed across various space nodes, and loosely connected to the enterprise service bus. While the space nodes per se are distinct from the ESB nodes, they are an incorporated part of the SOA4All DSB and exposed via the general the DBS access patterns (cf. Deliverable D1.4.1A).

A principle goal of the SOA4All DSB is to ensure that the SOA4All framework scales to the dimensions of the Web, by enabling appropriate distribution techniques that evolve the

traditional ESB techniques towards a fully Distributed Service Bus, without altering the communication and interaction patterns of the ESB core. While the state-of-the-art ESB technology relies mostly on client-server communication, the DSB provides SOA4All services with more powerful communication patterns (e.g., publish-subscribe, event-driven, semantic-oriented) that allow further decoupling of the communicating entities in terms of time, processing flow and data schema. In particular, the integrated support for semantics thanks to the Semantic Space infrastructure empowers direct links to reasoning or mediation techniques. By adding Semantic Spaces, SOA4All realizes an infrastructure that grants access to distributed semantic repositories for service descriptions, a storage infrastructure for sharing monitoring data and other collaborative tasks, and asynchronous and/or event-driven communication without requiring users to take care of additional access points. In other words, the SOA4All DSB provides the core infrastructural services of SOA4All in an all-in-one solution. This is an important principle for the scalable SOA4All Runtime, as it allows any type of communication efficiently and transparently over the Web by means of sharing or exchanging any type of data in between any type and number of distributed actors.

In summary, the SOA4All Distributed Service Bus exposes the traditional enterprise service bus functionality extended with scalable Web-style publishing and reading, integrated support for semantics and event-based communication mechanisms as basis for shared data management and collaborative activities.

2.1 Comparison and Contrast with the TripCom Project

TripCom builds upon earlier coordination and shared-memory systems such as Linda tuple spaces [1]. It enhances the principles underlying these systems with Web and semantic technologies in order to achieve a level of reliability, scalability, and Web-compatibility that is expected on today's Internet. From a conceptual point of view, TripCom advances tuplespace computing in two respects: 1) its coordination primitives are tailored to the Semantic Web, and 2) its distributed space structures are compatible with the fundamental Web principles, enabling for the first time a tuplespace infrastructure to reach global scale.

The coordination primitives of TripCom were defined with focus on the Semantic Web, but they were nevertheless also designed to be backward compatible with traditional Linda-based systems. TripCom defines various operations for writing, reading or removing RDF statements, corresponding to the core Linda operations. Additionally, TripCom includes a publish/subscribe mechanism that enables event-based service interactions.

From a user's point of view, the global Triple Space infrastructure is realized by means of virtual spaces and subspaces, called triplespaces. The TripCom space model allows the spaces to be organized in a forest (i.e., a set of non-overlapping trees), where every space can have at most one parent, but as many child spaces as necessary. This subspace model allows applications to structure the published data and to restrict access to the spaces with individualized access rules. For this, TripCom defines a sophisticated role-based security framework. While the security framework shows well both the power and the limits of security in a Triple Space, it is obviously a limiting factor especially when it comes to scalability. For this reason, TripCom had to step back from applying security also in more distributed settings described below, where scalability was of higher importance.

The well-conceptualized coordination primitives and the space model allow TripCom to be the first large-scale tuplespace installation. TripCom published very promising results with respect to the distribution of triplespace hierarchies across multiple hosts: the distribution mechanisms of TripCom support the discovery of spaces located on other nodes. In contrast, the project delivers limited results in distributing individual spaces and the contained semantic data across multiple triplespace nodes; individual triplespaces must be stored on single (server) nodes. In fact, TripCom's Triple Space realization is a minimal extension to a client-server implementation. Still, TripCom abstracts from the client-server characteristics of

its implementation. Thanks to the standardized coordination primitives and built-in space discovery mechanisms, triplespaces can be attained for reading or publishing from any access point (kernel) as one big global shared memory space.

In addition to the core space infrastructure, TripCom established a framework for binding the communication of WSDL/SOAP-based Web services to Triple Space. The project produced a sound and well-received showcase on how WSDL/SOAP can be used on top of Triple Space as an alternative to traditional HTTP-based interactions. In those examples, WSDL descriptions and SOAP messages are transformed into RDF and published, shared and read from the Triple Space.

As stated, TripCom yielded significant work in terms of the conceptualization of Triple Space as a global knowledge infrastructure. The project presented interesting results towards a large-scale virtualization layer for coordinating distributed repositories, and for coordinating distributed Semantic Web applications. As such, Triple Space has proven to be an enabler for collaborative activities of autonomous applications, as the project eventually illustrates by integrating Triple Space with WSMX. However, the TripCom prototype suffers from problems with performance and simplicity of implementation. In order to highlight the applicability of the tuplespace principle for component coordination, TripCom has chosen to use an open-source JavaSpace implementation for the integration of the various internal components of a kernel. While this simplifies the integration process, the loosely coupled realization significantly increases the complexity of the data flows, which results in observable performance loss at the level of individual kernels. Other implementation aspects, however, show applicability to Triple Space infrastructures such as namely the principle of so-called Triple Space kernels that in P2P-fashion provide the space functionality: persistent storage via OWLIM bindings, query processing, distribution (PGrid-based indexing), and security and trust. Still, the lack of performance and scalability disadvises to rely on reusing the TripCom implementation.

With respect to evaluation, TripCom has two deliverables that are dedicated to the investigation of scalability-related aspects at the level of specification, architecture and implementation. Moreover, a parametric-design-based requirements analysis tool was developed in order to support scalability-driven user requirements analysis and space development. Finally, the project conducted an experimental evaluation of Triple Space via a large-scale deployment on Amazon Elastic Computing Cloud (EC2). The rather limited results that could be achieved confirmed the usability and applicability of Triple Space to the (Semantic Web). In order to gain more insight into the scalability of Triple Spaces in large-scale scenarios, more in-depth investigation would have been necessary. Such evaluations could however not be done in course of the project due to a lack of resources and time.

To summarize, TripCom achieved significant and mature results at the level of specifications, also with respect to scalability and usability investigations in the context of service architectures. However, it lacks reusable results in distribution of triplespaces and in service coordination support at the level of the Triple Space. SOA4All will have to further investigate these aspects.

2.2 Semantic Space Architecture Specification

The principal requirements for the SOA4All Semantic Space infrastructure are distribution, openness and scalability. At least the distribution feature and as natural consequence there of, the scalability feature are the major drawbacks of the available TripCom approach. As Section 2.1 has discussed, TripCom provides distribution, but only in the sense of server distribution, and distributed hosting of individual spaces. In order to provide a fully distributed, globally accessible and scalable approach to Semantic Spaces, it is crucial to enable means that also distribute the data of individual spaces. Otherwise, we are limited in size and dependability to the limits of individual server nodes.

As a result of this, SOA4All exploits peer-to-peer technology to distribute the triples that are published and stored in the Semantic Space infrastructure. Details on the peer-to-peer approach and algorithms chosen are given in Section 3.1. At this stage of the deliverable, it is only important to recognize that the Semantic Space infrastructure is layered on top of a scalable peer-to-peer network (Figure 2).

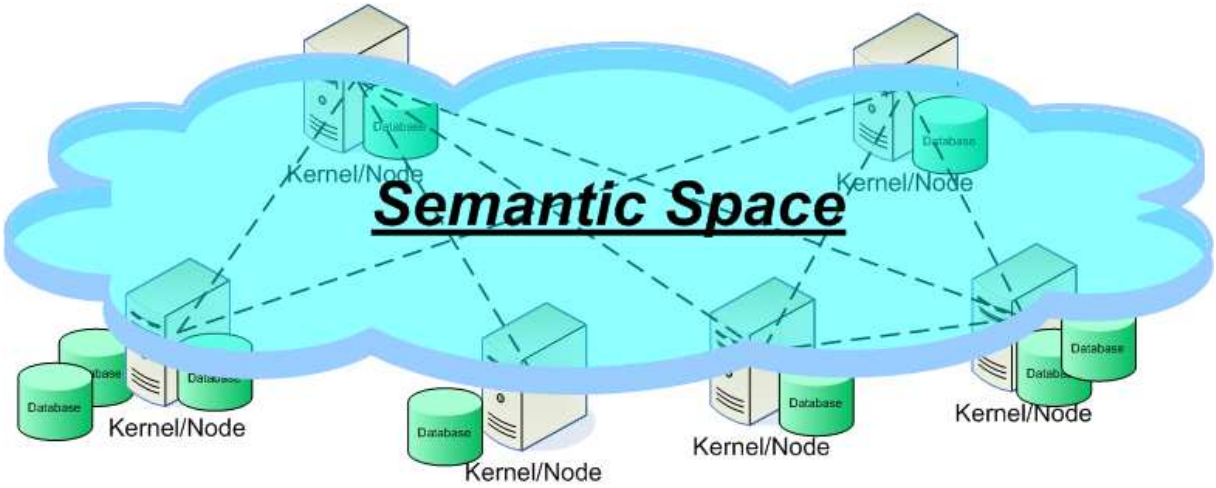

Figure 2: High Level View on Semantic Space Architecture

In SOA4All, we will enhance every node of the P2P network, also called kernel, with a local RDF repository for optimized triple storage, processing and query resolutions. The repository that will be used in SOA4All is OWLIM, which provides the Semantic Space infrastructure with inherited OWL-Horst reasoning with respect to the locally stored data of each node (http://www.ontotext.com/owlim/). At this stage the Semantic Space infrastructure does not provide any type of distributed reasoning, nor does the project aim for such solutions. In other words, the expressiveness of the query resolution mechanism is given by the expressiveness of the OWLIM query engine of each node hosting the RDF data. This does of course not exclude further processing of query results via the merging of result set or the intelligent distribution of queries across multiple nodes. Note however, that such an approach limits the constraint handling and join-capabilities of the query resolution algorithms, compared to federated repositories. At a later point in time, we plan to consider the exploitation of research results from projects like LARKC that investigates solutions that enable distributed reasoning over large data sets (www.larkc.eu). Work in that direction will however earliest be followed for milestone 3, and the second release of the Semantic Space architecture specification in M24 of the SOA4All project.[1]

In the continuation of this section, we focus on the architecture of the Semantic Space infrastructure that is layered on top of the peer-to-peer network. As Figure 2 depicts, the Semantic Space is provided by the sum of all kernels, where each individual kernel provides locally the functionality of the infrastructure. In turn, the kernel is the sum of all software components that are necessary to realize a global Semantic Space infrastructure. The set of components necessary is comparable to the ones established within the TripCom project: interface implementation, distribution manager to handle the P2P network connectivity, metadata management, query processing and certainly storage. In alignment with the rest of the project, there are no security considerations taken into account. Consequently, all spaces

[1] Several dedicated workshops for the exploitation of synergies between SOA4All and LarKC are planned and taking place soon. The alignment of the data layers is of particular interest.

are in principle open to the public, and issues like authorization, access restrictions, and secure communication are neglected throughout the discussion of Semantic Spaces. In fact, as research in TripCom has shown, in various aspects, the incorporation of security mechanisms clearly hampers the scalability and distributedness of the infrastructure, as security always requires a given degree of centralization. This was as such also discussed in [2]: it is argued that a system cannot ensure scalability, functionality and security the same time; i.e. only two out of three of these properties can sufficiently be taken into account. SOA4All concentrates on functionality for large scale service-oriented computing, and in this context, scalability is certainly the main challenge to take care of.

The Semantic Space infrastructure will be tightly bound to the core infrastructural service of SOA4All that is mainly implemented by the Distributed Service Bus (DSB). In short – for more details please consult Deliverable D1.4.1A – the DSB is an extension to the available open-source PeTALS ESB. PeTALS is a decentralized service bus realization that is put in place by a distributed set of PeTALS nodes. In that sense, there is a conceptual similarity observable between the approach chosen for the bus, and the approach presented here for the space infrastructure. While ESB nodes and space nodes can be co-located on the same provider machine, the two systems are distinct implementations and loosely coupled via connector components of the bus. In other words, the physical co-location is a design decision that is not a requirement. However, and this is important to note, within SOA4All, the Semantic Space is accessed via DSB binding components (EJB BC) only, and hence becomes a part of the core communication and coordination facility of SOA4All.

In the continuation of this section, we present the data and space models, the coordination interfaces and access patterns. Moreover, we introduce the different components that are necessary to realize the space functionality and their relationships.

2.2.1 SOA4All Semantic Space Data Model

The data model supported by the SOA4All Semantic Space infrastructure is the Resource Description Framework (RDF, [9]). RDF is the de facto standard for the representation and formalization of information about resources in the World Wide Web, and is particularly intended to be used for the representation of metadata (in the case of SOA4All: monitoring data, service and process description or user profiles). RDF is based on the principle that all resources have unique identifiers (in form of Uniform Resource Identifiers (URIs)), and that resources are described via properties and property values. In this way, RDF can represent simple statements about resources as graphs of nodes and arcs that represent the resources, their properties and values. With this concept in mind, RDF defines so-called RDF statements to consist of a subject, a predicate, and an object: t = <subject, predicate, object>, and an RDF graph to be a set of statements.

This notion of RDF, triples and graphs, has proven to be sufficient in past Semantic Space projects, and is expected to satisfy the requirements of the various SOA4All Platform Services and application scenarios of SOA4All too.

While RDF is used as the standard data model for the Semantic Space infrastructure, the SPARQL query language, and the simpler triple pattern concept are used at the level of retrieval. SPARQL is a standard SQL-like query language for RDF repositories that is based on triple patterns to describe the RDF statements to be match during the query process [10]. Triple patterns are RDF triples except that any of the subject, predicate and object fields can take the form of a variable. A SPARQL query thus contains a set of triple patterns that matches a subgraph of the RDF data in the repository.

2.2.2 SOA4All Semantic Space Model

The SOA4All Semantic Space model for spaces knows not only one big space for the sharing of semantic artefacts, but divides the overall infrastructure into many – administrator created spaces (cf. management operations a bit later in this section). The individual spaces

are a priori disjoint, and the publication processes as well as the retrieval processes are targeted at a particular space by means of the space's unique identifiers.[2] The use of unique identifiers (URIs) for spaces makes them addressable resources too, and spaces can also be annotated with metadata; i.e., the infrastructure can store and process knowledge about spaces, just as it processes data about other resources.

The specification of spaces as storage units at the level of the Semantic Space allows for the creation of application-specific storage platforms with individualized properties on topic, size but also with respect to non-functional features such as consistency, completeness or availability. This increases not only the perceived service that a space provides, but also the scalability of the overall infrastructure – as fewer actors interact and exchange smaller amounts of data over identifiable subunits of the Semantic Space infrastructure [7].

The implementation of virtual spaces has some implications on the way the semantic data is managed in the Semantic Space. A particular RDF triple (i.e., the same combination of subject, predicate and object) can be stored in multiple spaces, and is treated as a different piece of information in each of the spaces it was published to. In other words, although the triple would represent the same arc in the global RDF graph of the Semantic Space – if all data would be considered to reside in one container – they are treated as distinct pieces of information and maintained multiple times, once for every space.

The flat organizational structure for spaces that was presented so far is of course rather limited in its expressiveness and hence in the way it can support more complex interaction patterns between distributed actors. SOA4All therefore adopts and improves the principle of space hierarchies that was first realized in course of the TripCom project. Any two spaces can be brought in relationship via meta-level links (one of the reasons for seeing spaces as resources) and more complex spaces and federations can be created on the fly (cf. Section 3.2). The most prominent relationship is the subspace-superspace link that allows the installation of space trees, in which the subspace is a part of the superspace – which also applies to the published data. In that way, the Semantic Spaces are organized in tree hierarchies. In fact, as there might be multiple such space hierarchies, the whole Semantic Space infrastructure delivers a forest of spaces. In SOA4All, subspaces are defined to be in a part-of relationship with their parent, and therefore the semantic artefacts published to a subspace are inherently content to the parent space. All queries expressed to a parent space are resolved against the data of the entire sub-tree unless a client explicitly states that non-recursive retrieval is required, i.e., retrieval that does not take into account the hierarchies.

As any space can be parent or child to any other space, and the subspace relations thus build graphs, care has to be taken with respect to cycles. Subspace relationships can therefore only be defined, in the case that the new link does not create a cycle, and the space model of SOA4All is determined by an acyclic directed graph.

2.2.3 SOA4All Semantic Space Access Operations

There are two categories of operations specified for the Semantic Space infrastructure. First, there are operations to install and configure Semantic Spaces, to join spaces and to provide statements about them (Table 1), i.e., to indicate links and participation in federations (cf. Section 3.2). The second type of operations exposes the space functionality to clients that have to publish to, read from and subscribe to particular spaces (Table 2).

Table 1: Management Operations:

createSpace(URI space, [URI	Enables the creation of a new space with the given

[2] Note that we also permit the invocation of a request without indicating a target space (cf. client access operations in Section 2.2.3).

parent]): void	identifier -- optionally as direct subspace of the indicated parent space.
joinSpace(URI space): void	Allows an administrator of an infrastructure node to join the network of nodes that manages the data of a given space.
leaveSpace(URI space): void	The inverse operation of joinSpace().
listJoinedSpace(): Set<URI>	Returns the identifiers of all the spaces that the local space node co-manages.

Table 2: Client Operations:

write(URI space, Set<Triple>): void	Enables the publication of the given set of triples to the indicated space. This operation is non-blocking and returns immediately, guaranteeing that eventually all the triples are persistently stored in the chosen space.
query(URI space, Query query): Set<Triple>	Enables the retrieval of matching triples that are available in the indicated space by means of a SPARQL CONSTRUCT query, or a simple triple pattern. The retrieval operation takes a best-effort approach, makes however no guarantees about the completeness of the result set. The only guarantee is that if there are matching triples available in the space, they will eventually be retrievable.
queryNonRecursive(URI space, Query query): Set<Triple>	As above, but without taking into account the space hierarchy; i.e. only the directly addressed space is queried.
query(Query query): Set<Triple>	As above, but the query is not resolved against the triples in a particular space, but against a large undetermined set of spaces; eventually all known spaces.
queryV(URI space, Query query): QueryResult	As above, but does not return triples but variable bindings; i.e. this variant support SPARQL SELECT queries.
queryVNonRecursive(URI space, Query query): QueryResult	As above, but without taking into account the space hierarchy; i.e. only the directly addressed space is queried.
queryV(Query query): QueryResult	As above, but does not return triples but variable bindings; i.e. this variant support SPARQL SELECT queries.
remove(URI space, Query query): Set<Triple>	Same as the basic query operation, however this operation does not only return the matching triples, but removes them from the given space.
subscribe(URI space, Query query, Listener listener): URI	Enables the subscription to a given space by means of a query that shall be matched. Upon the discovery of new matching content, the listener is informed about the recent updates. Every subscription is associated with a unique identifier.
unsubscribe(URI subscription): void	The inverse operation of subscribe: unsubscribe removes the subscription that is identified by the provided URI.

In order to have a clear and transparent information flow between the Semantic Space and the space users in case of troubles, there is a set of exceptions and error messages defined:

- ParserException: indicates that there was a problem with the triple serialization at publication time, or the query syntax at retrieval time.
- SystemException: indicates that there occurred some unexpected problem in the space infrastructure.
- SpaceException: indicates that there was a problem with the space management
    - SpaceAlreadyExistsException: indicates that the space that shall be created already exists in the system.
    - SpaceNotKnownException: indicates that a user likes to publish to or retrieve from a space that is not known to the system.

2.2.4 SOA4All Semantic Space Kernel Architecture

The functionality of the Semantic Space infrastructure is realized by a number of distributed pieces of software that we term 'kernel'. A kernel is the sum of components that jointly provide the local realization of the space platform, while all kernels that share a space provide the infrastructure in collaboration. Five main components constitute a Semantic Space kernel (Figure 3):

- interface implementation,
- distribution manager to handle the P2P network connectivity,
- metadata management for the provisioning and processing of structure information about space federations and organization (cf. Section 3.2),
- query processing to optimize the query processing on top of the P2P network, and
- the storage component, which grants local access to the RDF repository.

We first shortly introduce the functionality of each of the five components in the listing below, before considering their interactions by means of two sequence diagrams (Figure 4 and

Figure 5). The figures indicate the role and relationship of the different components in course of publication and retrieval from a given space.

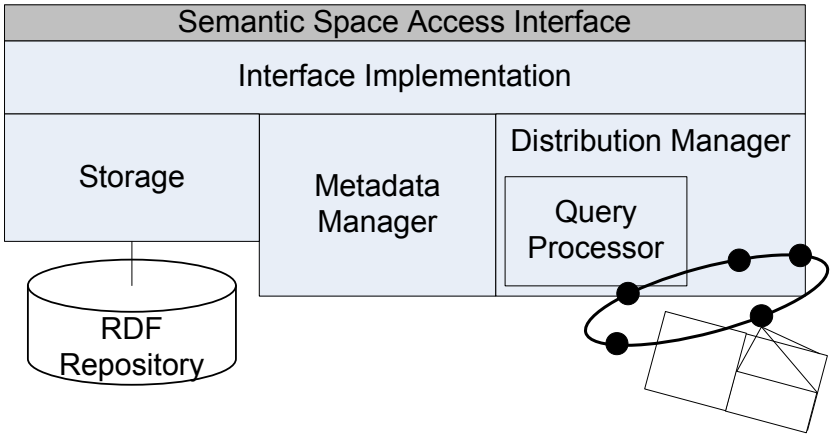

Figure 3: Semantic Space Kernel Architecture

Interface Implementation: The task of the interface implementation component is to expose the SOA4All access operations to clients, and to trigger respective actions in the kernel, and

likely on other kernels too. The implementation is expected to be rather simplistic, as the actual functionality of a kernel is provided by the sub-mentioned four components.

Distribution Manager: For most of the distribution manager, the reader is referred to Section 3.1 that describes in detail how the data is distributed across the various nodes of the Semantic Space, and how queries can be resolved in such an environment. Besides enabling the indexing infrastructure of the P2P overlay, the distribution manager component is responsible for the communication to and coordination with other kernels and the network.

Query Processor: The query processor is closely related to the distribution manager. As Section 3.1 will show, the Semantic Space relies on a two-layered indexing system, with one layer for the management of spaces, and the second for the management of the application data. It is possible that a given query must be propagated to various nodes within the global network of Semantic Space nodes, and finally, that results from distributed nodes must be merged. As mentioned above, each node that shares a given space has its own query engine that resolves user request against the locally stored data, and the results are gathered by the kernel that handles the retrieval operation.

Note that the query-processing component is tightly coupled to the distribution manager, as optimizing query and answer processing depends on the underlying P2P implementation. In fact, it is very important that data distribution (storage) is going hand in hand with data discovery (querying) in order to optimize the performance and scalability of the infrastructure.

Metadata Manager: The metadata manager is aware of all Semantic Space-related schemas and is able to provision, process and present metadata independently of the user-driven processes, such as accessing the storage or accessing the P2P network. As it was presented in the paragraph about the space organization, the Semantic Space offers the possibility to structure spaces, to create federation and consequently to combine and separate space structure on a per application basis, or even on a per operation basis. The structural metadata that is required to realize such flexibility in space organization is the main target of the metadata manager. The structural metadata is shared amongst all space nodes in an own global metadata space. An additional possibility is the collection of access information metadata. Such metadata would enable the access pattern-driven (frequency, relation and recency of access) organization of data distribution algorithms, the discovery of related artefacts, or the management of caching mechanism. As past projects have shown, there was so far no scenario realized, which would convincingly argue for the added overhead. Therefore, this second option will not be pursued for the time being.

Storage Component: The storage component is a connector to an OWLIM RDF repository. Within SOA4All, there will be no research on RDF storage performed, and the kernel makes use of OWLIM and its query engine support that is sufficient for the project.

In the following, we provide two sequence diagrams: Figure 4 for publication, and

Figure 5 for retrieval from a space. A third sequence diagram depicting the execution of retrieval from an implicit federation is given in Annex A (Federations are subject to Section 3.2).

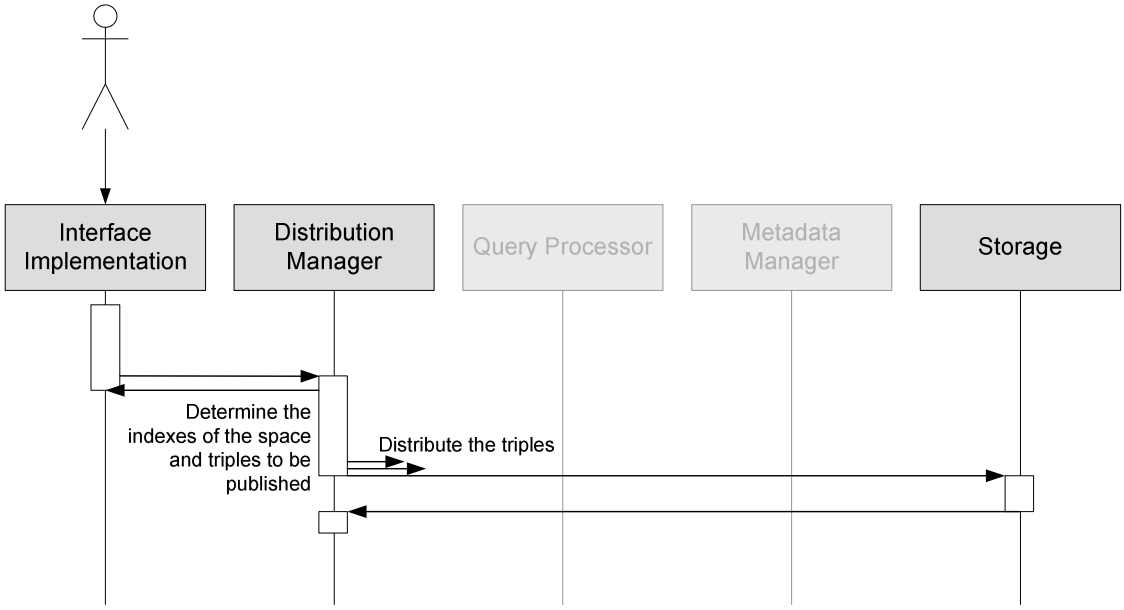

Figure 4: Sequence Diagram of the Publication of Triples to a Given Space

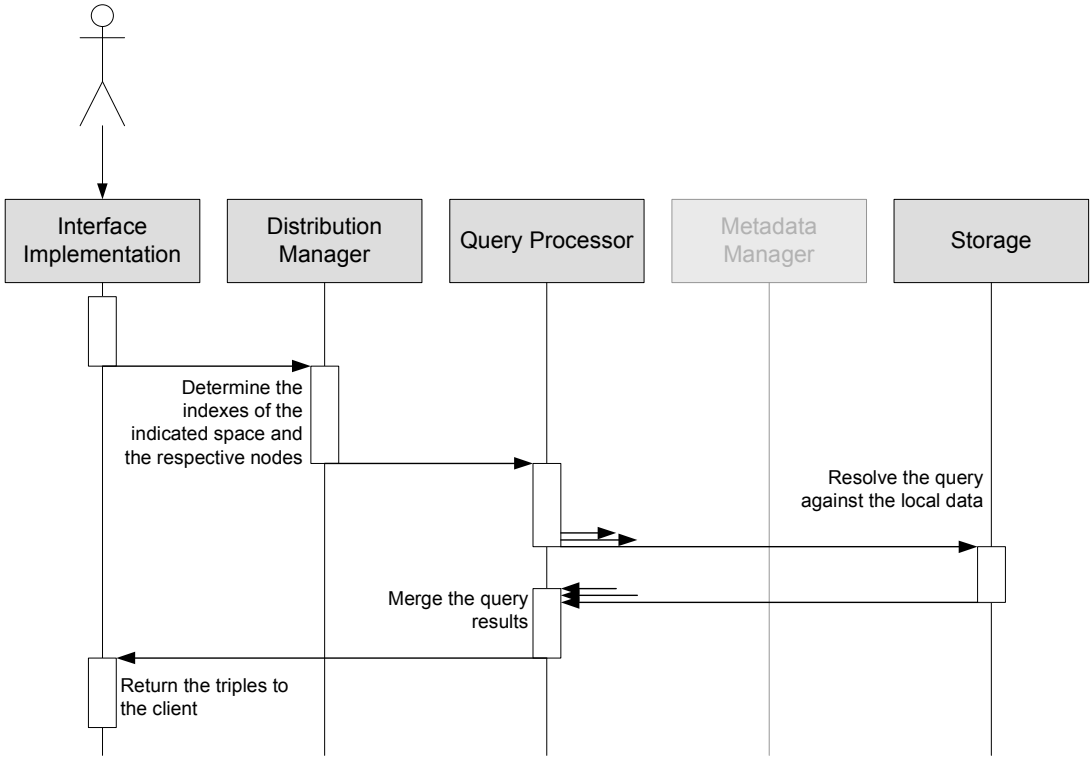

Figure 5: Sequence diagram of the retrieval from a given space

3. Distribution and Federation

This section describes the use of P2P solutions for building a large-scale distributed Semantic Space infrastructure, which includes data storage and query processing. It also articulates the federation concept aiming to integrate and combine data stored in different spaces.

3.1 Scalability Through Distribution

Scalability is an utterly important feature in SOA4All. P2P networks have often been considered as the ultimate solution to ensure this feature thanks to the cooperative environment that could be built based on a set of techniques. The focus of this section is to present the P2P architectures that will be used to build our scalable and distributed Semantic Space infrastructure.

3.1.1 Peer-to-peer Overlays

Because of traditional problems of centralized systems such as performance bottlenecks, fault-tolerant issues, P2P architectures have been considered as promising solutions to reach high scalability. In principle, P2P architectures do no have a central control unit or hierarchical organization: each peer is equivalent in functionality and cooperates with other peers in order to solve a collective task such as distributed storage or query processing. Unlike centralized architectures, P2P techniques generally prevent bottlenecks such as traffic overload on a central server.

As mentioned in Deliverable D1.3.1, P2P systems are classified into two basic architectures: structured and unstructured overlays. Unlike unstructured P2P systems, structured overlays are mostly based on Distributed Hash Tables (DHTs) where peers form a deterministic graph. The success of DHTs is bases mainly in their scalability guarantees. As it is shown in [8], using an efficient routing strategy, DHTs provide a routing performance of *O(log N)* in a network of *N* peers. For scalability reasons, we have considered P2P technology for data storage and retrieval in the context of the SOA4All project.

The P2P Semantic Space infrastructure is based on structured overlay networks. For better understanding, we briefly introduce some existing work that we will adopt for the infrastructure design. DHT-based systems provide the following basic functionalities: store (key, object) storing an object identified by its key, and lookup(key) which returns the object (when it exists) from the peer that is responsible for the key.

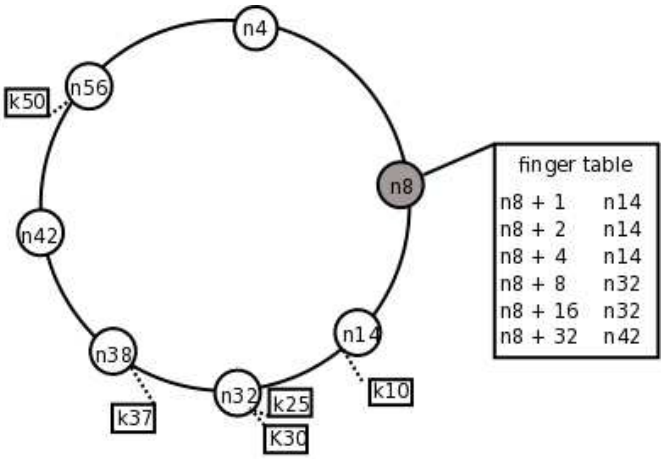

Figure 6: Chord ring with peer IDs and data keys (Finger Table of node n8)

Chord is a structured P2P overlay network based on Distributed Hash Tables [14]. Peers are organized in a logical ring topology. Chord uses consistent hashing to map peers into an *m*-

bit circular identifier space. The main property of the consistent hashing is ensuring load balancing between nodes, i.e., with high probability, peers receive roughly the same number of keys. Each data item that is stored in the network has a unique key identifier. Data discovery can be easily implemented on top of Chord overlay by storing the *(key, value)*-pair at the peer to which that key maps. So that each retrieval operation is forwarded to the nearest node to the data location until the data unit is found. Moreover, each node maintains a routing table with (at most) *m* entries called the finger table. It contains IDs of a set of nodes around the ring overlay. When receiving the query lookup(key), a peer forwards the query to a node in its finger table with the highest ID not exceeding the value hash(key). Figure 6 illustrates a Chord ring overlay with six nodes and five data keys as well as the finger table of node n8.

CAN is another DHT-based P2P overlay, where the architectural design is a virtual multi-dimensional coordinate space [11]. This Cartesian space is dynamically partitioned among all nodes in the system such that each node “owns” a zone in the overall space. Figure 7 shows a 2-dimensional CAN space partitioned between seven CAN nodes. This virtual coordinate space is used to store *(key, value)*-pairs. For instance, to store a *(key, value)*-pair, the key *k* is deterministically mapped onto a point *p* in the coordinate space. The *(key, value)*-pair is then stored at the node that owns the zone in which the point *p* lies in. The lookup process of a value *v* corresponding to a key *k* is achieved by applying the same deterministic function on *k* in order to map it into *p*. The query for a given value will be routed through intermediate nodes in the overlay until it reaches the peer's zone containing *p*. A peer joining the network corresponds to randomly picking a point *p*’ in the Cartesian coordinate space and routing to the zone that contains the point. A zone will be allocated to the new peer by splitting the current peer owner zone in half: keeping half for the original peer owner and allocate the other one to the new peer. Figure 7 (right) shows the zone splitting process after a join by peer *j*. Similarly, when a peer leaves the network, zones can be merged and peers' neighbour lists will be updated accordingly.

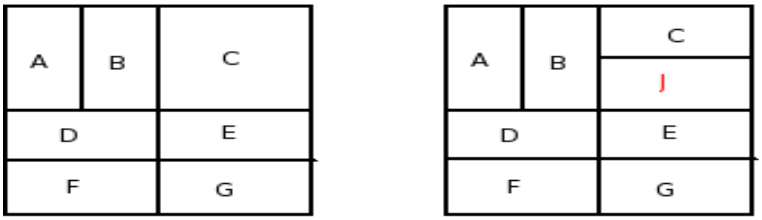

Figure 7: Peers in a 2D-CAN Overlay Before and After the Join of Peer j

The motivations behind using structured overlays in the context of SOA4All Semantic Space infrastructure are the following:

- Ensuring a scalable infrastructure for storage and retrieval of RDF data. Data can be stored on specified peers so that messages and/or data can be efficiently routed to the responsible physical location
- Search complexity is guaranteed to be logarithmically in (most) DHT-based P2P overlays

The Chord and CAN architectures discussed above present the Semantic Space infrastructure key design principles. Indeed, the overall space infrastructure is organized as a multi-layers P2P network, as shown in Figure 8. More precisely, the lower level layer consists of a set of 3-dimensional CAN overlays. Each CAN space is connecting a set of Semantic Space kernels that store triples belonging to the same Semantic Space. In other words, one Semantic Space can be managed by more that one peer (kernel), and the data that is shared in such a space is also shared in the given CAN overlay. The three dimensions of each CAN coordinate space represent respectively the subject, the predicate and the object of the

stored RDF triples. The size of each peer's zone reflects the amount of RDF data that it maintains.

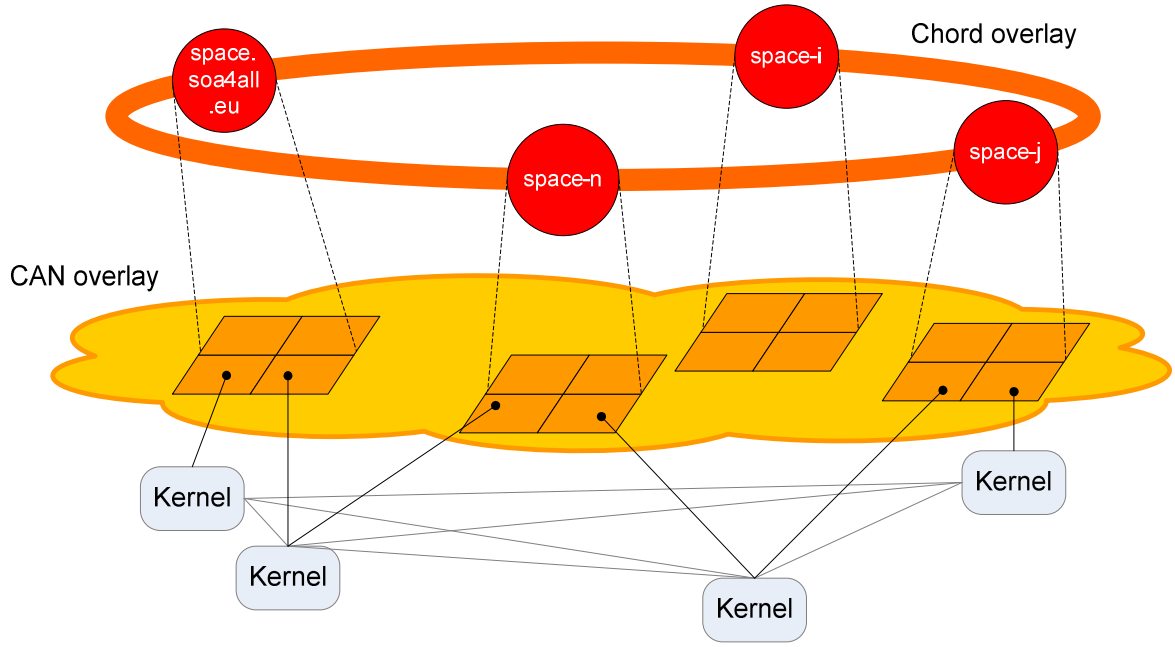

Figure 8: P2P Semantic Space Infrastructure

The upper layer of the Semantic Space storage infrastructure is a Chord-based ring overlay on which the different Semantic Spaces are represented. Each node of the Chord overlay represents a Semantic Space and maintains references to various peers of the CAN overlay.

Since each Chord node requires a unique ID, we make use of the unique identifier *spaceURI* that is associated with each Semantic Space. Consequently, the data routing process runs in two phases: i) routing to the appropriate space at the Chord-ring by means of the spaceURI; ii) once the space is discovered, the RDF data is routed towards the right node in the CAN layer.

3.1.2 Data Storage

As it is stated in Section 2.3, the data model of the SOA4All Semantic Space infrastructure is the Resource Description Framework (RDF) where a RDF triple is composed of a subject (s), a predicate (p) and an object (o). We also consider that a Semantic Space is identifiable by a *spaceURI*. The *spaceURI* is used to route data and/or queries at the Chord level, whereas the RDF triple resources help to find the relevant peer to store the triples in the CAN overlay. In the CAN layer, a lexicographical order is used to find the position in the P2P coordinate space where a triple will be stored. This helps to efficiently manage range queries during a search process. For instance, considering the following example for the data item *(s,p,o,spaceURI)*: first, a key *ID* is computed by applying a hash function to the *spaceURI*. This *ID* is used to route along the Chord-ring; second, the *(s,p,o)* resources are used to determine the coordinates in the CAN overlay space. Thus, the point *m* whose coordinates in the CAN are (*s,p,o)* refers to a point, which is under the management of a peer *n*. This peer will store and manage the triple.

3.1.3 RDF Querying Process

In order to efficiently process queries in a large scale distributed environment, sophisticated mechanisms are required. The two-layer indexing architecture of the SOA4All Semantic Space infrastructure supports the scalable processing of the following types of queries:

- Atomic queries are triple pattern-based where the subject, predicate, and the object can be either variables or accurate values. As an example, one could look for a triple

with a fixed subject value but without any predicate or object specified. In such a situation, the query will be routed on only one axis of the CAN overlay. Once a peer is found, it forwards the query only to its neighbours that are most likely "in the direction" of peers storing the corresponding triples based on the peer zone's coordinates. For instance, for a query *Q = (<s><p>?o)*, the retrieval process must return all objects satisfying the subject s and the predicate p. The resource s gives the coordinates of all peers according to the subject axis in the CAN overlay that stores triples for the subject s. Similarly, a search for p along the predicate axis returns peers storing the predicate p. The result of the query is the intersection of the two previous peer lists.

- Since Conjunctive queries are expressed as a conjunction of a set of atomic triple patterns (sub-queries), atomic triples will be processed first. Their result set will be returned and the intersection will be created. The details regarding this process are given bellow.
- Range queries can be efficiently supported thanks to the preservation of lexicographic ordering in the CAN overlay adopted by the storage process. As an example, we consider the following query *Q = (<s><p> ?o FILTER ($v_1 \leq ?o \leq v_2$))* with a given subject and predicate. The query seeks a set of objects, given by the variable ?o, in the range $[v_1,v_2]$. The routing process starts by finding the subject and the predicate fixed by the query. After that, it locates the lowest value in the range by going over the object axis. If all results are found locally, they are returned to the requester. Otherwise, the query is forwarded to neighbours that may contain other potential results.

The querying process can be performed with or without *spaceURI*. At the Chord level, if the *spaceURI* is given, the query will be routed to the relevant Semantic Space node. From there, the query is routed to the appropriate CAN space. If no *spaceURI* is available, the query is flooded at the Chord layer and the same routing process will be performed non-exhaustively at each CAN space. For performance reasons, the range of the query flooding mechanism will be limited with a time-to-life, respectively a hop count. The value of this threshold is subject to investigation in course of the implementation.

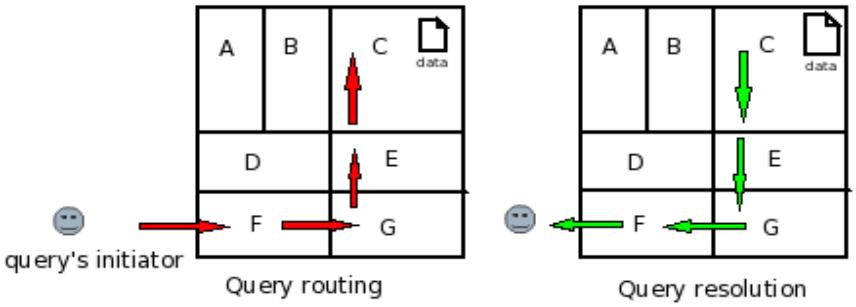

Figure 9: Query Routing and Resolution Inside a Space

At the CAN level, each query on a space will boil down to querying one or many peers. However, the caller should not be exposed to the implementation details and thus should not have to deal with peers directly. To achieve this transparency, we will use the following mechanism. The caller will contact a randomly chosen peer and give it the query. This peer will be called the *originator*. If the peer cannot process it because some data are available on other peers, then it will forward the request using the CAN routing scheme. For example, we assume a conjunctive query *q* to consist of a set of triples patterns q_1, q_2… and q_n. We suppose that each triple pattern will be resolved by a different node. The results will also use the same routing to come back to the first contacted peer and will be transmitted back to the caller. An example is provided in Figure 9 in which a request is sent first to peer F. Since the requested information is stored on peer C, the request is forwarded there and the result is then routed back to F, which transmits it to the initiator.

To query data from multiple peers, we apply the same algorithm, as for the querying of a single peer in the CAN overlay. The results coming form peers involved in the query resolution will be sent back to the originator, which will perform the merging before sending it to the initiator. It is possible to query multiple spaces explicitly by means of multiple space URIs or by flooding the Chord overlay to propagate a query to all spaces. If an administrator has defined a subspace relation, then multiple spaces will be queried implicitly; the same counts for federations in general. The following mechanism will be used

- Upon reaching the first space, the query will be passed to an *originator* peer which will treat it in the current space
- The originator peer will propagate the query to subspaces (or spaces in the federation) where they will be processed
- The originator peer processes the results from all queried spaces in order to provide the global answer set.

3.1.4 Scalability

The P2P architecture will naturally provide scalability regarding the storage space. As more peers are added, the space will be able to store more RDF data. However, this will have an impact on the performance of request/write operations.

First, consider write operations or requests processed by a single peer. The performance of these operations will mainly be limited by the routing from the originator peer to the target. In CAN overlays, the complexity is $O(dn^{1/d})$ where n is the number of peers and d the dimension. Since the dimension is fixed, the only way to lower the complexity is to maintain a low number of peers.

On the other hand, if a request has to be processed by multiple peers then the limiting factor will be the transmission time and processing time of all peers involved. Indeed, the originator will have to wait for all intermediate results to be available before performing the merging and sending back the matching set. There are known solutions to tackle these kinds of issues. First, a peer will have the possibility to batch or merge queries. When it receives a request that needs to be sent to another node, it can wait for a short duration for other related requests. If there are any, batching (send multiple requests in one single remote call) or merging (if two requests overlap, then only perform one) can be applied. Second, a peer could cache the results of queries processed by other peers and return them when similar requests are performed. Finally, the implementation will parallelize communication and computation so that a peer can compute its own queries; send others to its neighbours and process results when they arrive simultaneously.

3.2 Integration Through Federation

In the previous section, we presented how scalability is achieved via distribution. By means of known P2P overlay technology, it is possible to distribute triples, and to discover the data in highly scalable manners. In this section, we focus on what we call federation of spaces. Federations help to integrate arbitrary sources, by combining the content of distributed Semantic Spaces.

An individual Semantic Space is a resource of the global Semantic Space infrastructure. As it was presented in Section 2.2, a space is envisioned to provide an a priori single purposed shared memory for semantic artefacts. Spaces are created at will in order to group semantic data, or the actors collaborating on top of it. Within the SOA4All Distributed Service Bus, there are for example default spaces for sharing monitoring data, and for incorporating service registries that are used by discovery services, composition services and alike. It is

unrealistic to assume that the whole world, as an example, will publish semantic annotations to the same Semantic Space. The creation of particular spaces allows for the establishment of isolated repositories on a per case basis. While the access methods, the connectors and protocols remain the same, the use of particular Semantic Spaces with each having its unique identifier permits for example to change the scope of a discovery process, without changing the discovery procedures.

Installing individual spaces per topic, per communication need etc. creates smaller parts and hence an overall more scalable approach. The same time however, it fragments the Semantic Space and querying individual spaces might not lead to the expected goal, as cross-topic, cross-domain request might not be answerable. In such situations, it is necessary to integrate the semantic data of different spaces; a process that we enable via federations. In the next paragraph, we will show how SOA4All enables the creation of federations, and before concluding the section, we will present the technologies and algorithms necessary to build federations and query them.

Creating Federations

Table 3 depicts two operations that enable the creation of an explicit federation, and that retrieve a unique identifier for it. A federation is a set of spaces that are put in relationship by temporarily creating a virtual super-space for the given set of spaces. Federations are temporary in the sense that a user can remove the federation at any time on his own will via the unique identifier of the federation (deleteFederation). Federations can, as long as they exist, be queried with the same operations as regular spaces, e.g., by means of query(space, query) or rather query(federation, query). The interface implementation makes no difference in the processing of retrieval request form a space or a federation. Note moreover, that federations are bound to the kernel on which they were created, and, in contrast to spaces, are no global data containers.[3]

Table 3: Management Operations for Federated Spaces

createFederation(Set<URI> spaces): URI	Creates a temporary federation by explicitly grouping the spaces in the set under a common umbrella.
deleteFederation(URI fed)	Deletes the federation

An alternative approach to explicit federations is the installation of links between spaces. Besides the part-of relationship between spaces in terms of subspace hierarchies, there are further relationships between spaces that enlarge the scope and content of the retrievable semantic data. The relations are specified by means of predicates that give a meaning to the connection of two spaces: <*space1 inSomeRelationship space2*>. The relationships form a graph of interlinked spaces that is given by structural metadata. The retrieval process can then follow these links to enlarge the search scope beyond the space that was explicitly addressed by the user, and we refer to the concept of implicit federation. In order to define the relationships between spaces, there is an operation offered to administrators (Semantic Space users that manipulate the structure of the system, rather than only data) to create the links via ontological statements (Table 4), and thus to create implicit federations.

Table 4: Management Operations for Space Relationships

createSpaceRelation(URI sSpaces, URI relation, URI oSpace): void	Creates an ontological link between the subject space (sSpace) and the object space (oSpace). The type of relation is given according to the structural

[3] This binds a federation in the more restrictive sense to a given application, the one that access the Semantic Space infrastructure via that one kernel, because federations are not published.

	metadata ontology, and is used to query the infrastructure by means of the operations that are depicted in Table 5.

The ontological relationships, applied as the predicates of structural metadata statements, are the following (partly derived from the TripCom project):

Subspace (a transitive property) – Subspaces define a meronymic relationship between two spaces. The subspace is a part of the parent space, and as such, from the parent's point of view an inclusive part of it. In consequence, the subspace is also transparent to users that access the parent space. Subspace relationships are always defined by administrators and (a priori) given at creation time of a space.

Superspace (a transitive property) – Superspaces define a meronymic relationship and describe the inverse of the subspace relationship.

Similar space (a symmetric property) – Similarity describes a relationship between spaces that could answer the same queries with semantically the same responses, i.e., the knowledge content of the two spaces is the same, while they are potentially quite different in terms of the actual triples stored. Similar spaces are clearly not replicas, but they describe the same subjects with the same ontological bindings. In other words, two spaces x, y are similar if they both contain triples t_j with j = 1...n about subjects s_k with k=1…m that denote the same resources r_l with l=1…m, and if they both apply a common set of predicates from the same ontology o to annotate s. The parameters n and m will have to be investigated in order to determine useful minimal values.

Affine spaces (a symmetric property) – Affinity denotes a link between two space that contain information of the same topic, i.e., that describe subjects of the same class types, or use properties of the same ontologies – except obviously RDF(S) and OWL vocabularies. Two spaces x, y are considered affine if they both contain triples tj with j = 1...n about subjects s of type c as defined in ontology o, or predicates of property p that are defined in ontology o. The number n of triples that are necessary to call two spaces affine must be determined.

Collective spaces – The collection property defines the most general type of relationship. The link does not imply any particular type of relationship, but provides a generic means to connect two arbitrary spaces.

This set of relationships is not closed, as further specific clustering predicates can be defined as simple extension of the ontological model given above. The simplest approach is to extend the generic property of collective spaces and to assign a well-founded meaning to it. Administrators will be able to define such extension via configuration and management operations on the Semantic Space.

The flexible Chord-ring approach to space discovery allows the SOA4All Semantic Space infrastructure to take any type of relationship into account. Federations can thus be given by means of metadata statements about the structure of the space organization, or via an extended set of client operations that take into account sets of space identifiers rather than single space URIs only (Table 5). While the relationships that are given by means of ontological links are permanent and visible to the world, as they are published in a shared metadata space, the approach by use of explicit federations is a temporary definition of a federation and is only valid for the lifetime of the request.

Table 5: Client Operations for Federated Spaces

query(URI space, URI relation, Query query): Set<Triple>	The same operation as 'query' in Table 2 enhanced with an implicit federation parameter. The relation

	property, chosen from the list above, in form of an RDF predicate is used to select the desired federation from the structural metadata.
query(URI federation, Query query): Set<Triple>	*Although the federation represent a list of spaces, this operation simply queries the source given by the URI, and is hence identical to query(URI space, Query query) in Table 2.*
queryV(URI space, RDFURI relation, Query query): QueryResult	As above but for SPARQL SELECT queries.
queryV(URI federation, Query query): QueryResult	*As above but for SPARQL SELECT queries.*

There are no operations defined to add or remove triples via federations, as federations behave like read-only views in database systems. Triples are always published to a given space and stored there, and hence have to be removed from a given space too. Federations do never actually contain triples, but only virtually unite the triples of the contained spaces. Thus, also when federations are put in place, the smallest management container is still the space. In summary, federations can only be used for operations that do not alter the state and content of spaces.

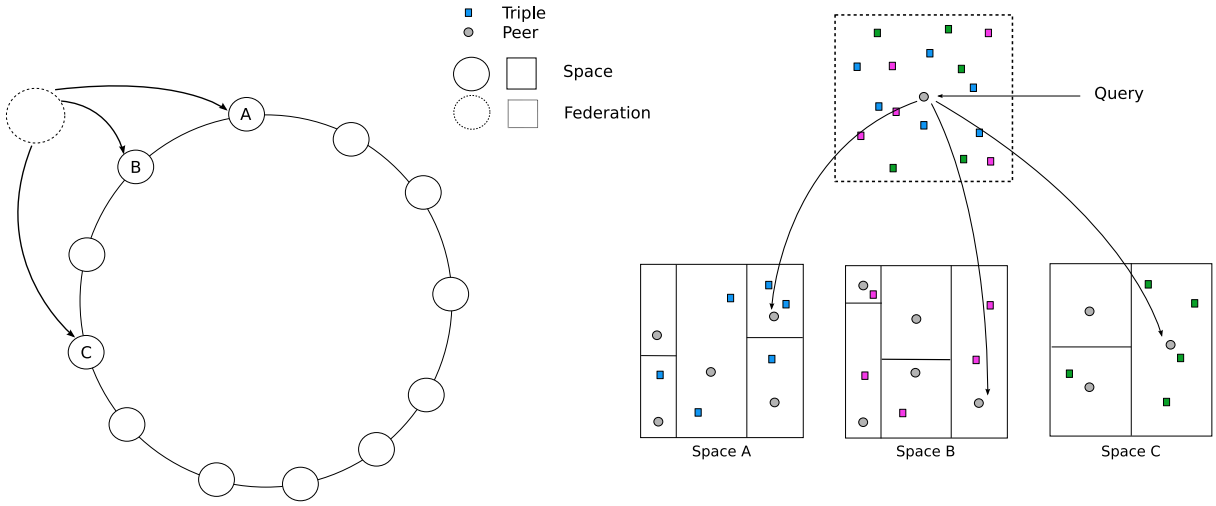

Figure 10: Federations at the Chord and Can Levels

Figure 10 shows the relation between a federation and the Chord/CAN overlays. On the left side, the image shows that a federation is outside the Chord but has references to existing spaces. From the user point of view (right side), it is a CAN space with a single (virtual) peer which only propagates the queries to all the federated spaces. The CAN overlay of federations does not index triples but simply stores references to spaces. When a query is performed on the federation, it is automatically propagated to all spaces involved and the resulting matches are merged.

3.2.1 More Efficient Implementation

The implementation described in the previous section has an important drawback, the lack of efficiency. Each query from a user will trigger requests, using bandwidth and resources of all the federated spaces although some of them might not be able to provide a match. We plan to address this performance limitation through different strategies.

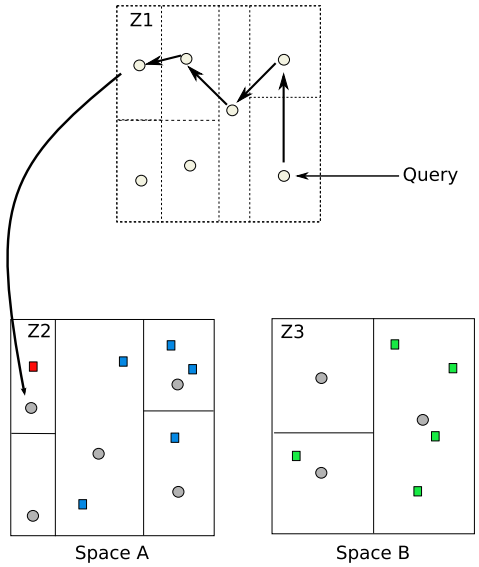

Figure 11: Optimized Query Processing in Federations

The first one will consist in having some caching mechanism in the federation, to lower the number of requests on the network. A second possibility will be the building of an optimized virtual CAN so that each query sent to the federation will only be propagated to a subset of the spaces likely to respond. More precisely, the query will only be sent to spaces, which might contain the requested triples and avoid those that currently do not store matching data (e.g., the requested subject is not indexed for the space).

Figure 11 shows an example of a possible optimization. The federation of spaces A and B is built by overlapping the different areas of responsibility of all peers. At the federation level, any triple falling into the area Z1 could be mapped into the areas Z2 or Z3. However, when building the federation, one can take into account that Z3 does not contain any triples. Therefore, if a user queries the federation for triples stored in the zone Z1, the request will only be propagated to Space A, avoiding unnecessary requests to Space B.

4. Publish-Subscribe

Publish-subscribe is an asynchronous communication paradigm where senders do not need to explicitly know the receivers [3]. Subscribers can express their interest in an event or a pattern of events and are notified of any event generated by a publisher that matches their interests. The events are propagated asynchronously to all subscribers. This paradigm provides further decoupling in time, space and synchronization between participants. First, they do not need to participate in the relation at the same time. Second, they do not need to know each other and can even be located in separate domains. Finally, processes are not blocked when generating events and subscribers can get the notifications asynchronously.

The publish-subscribe paradigm provides many different ways of specifying interests. The main two are topic/subject-based and content-based. In the topic/subject-based approach, participants can subscribe to individual topics, identified by keywords. Topics can be organised into hierarchies, based on containment relationships. Subscriptions to a topic will trigger subscriptions to all subtopics. In content-based subscriptions, it is possible to express an interest based on properties of the events, such as their attributes.

4.1 Application to SOA4All

The publish-subscribe mechanism of SOA4All allows users to be notified when some particular triples are added to a space. Topic-based subscription will be implemented at the granularity of a space. A user will be notified if any triple is added to its space of interest or any of the related subspaces.

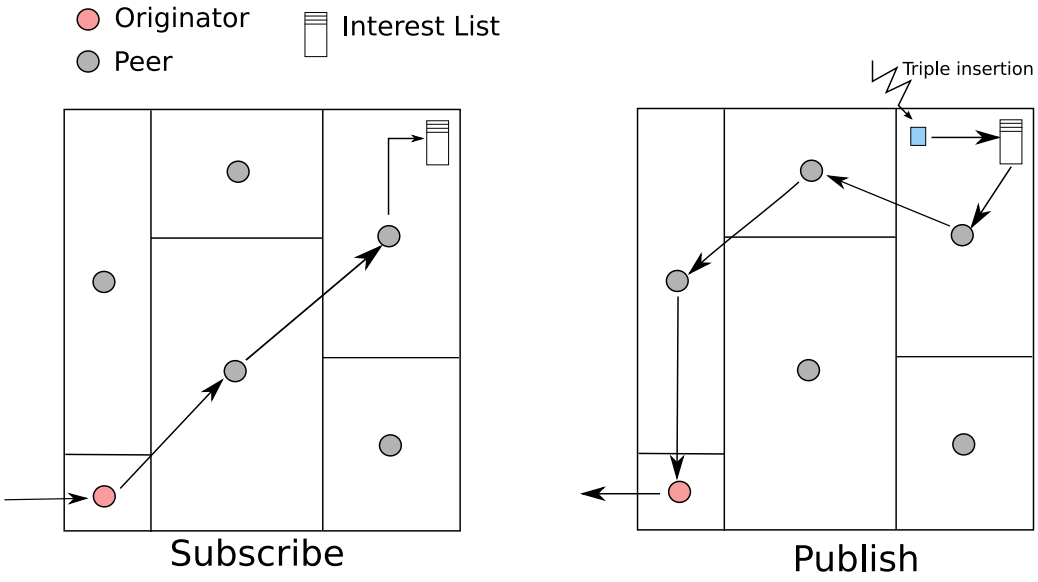

Figure 12: Content-based Publish-Subscribe in a Space

If finer grained notifications are needed, we will provide a content-based scheme, using RDF queries. The user will provide a RDF query when subscribing. If a matching triple is added to the indicated space, an event will be generated.

An example of the content-based subscription mechanism is presented in Figure 12. A user first contacts a randomly chosen peer in the CAN, which then acts as originator peer. The subscription will first be routed like a standard query. When reaching a peer able to process it, the query will be stored in a local table, called interest list, along with references to the originator peer. When a triple will be added, the query will be executed against it and if it

matches, an event will be generated back to the originator, which in turn will transmit back to the user.

Table 6 depicts the different operations that can be performed by the user in order to manage subscriptions.

Table 6: Management Operations for Subscription

subscribe(URI space): void	Subscribe for all events in the space (topic-based)
subscribe(URI space, Query query): void	Subscribe for all events matching the provided query (content-based)
unsubscribe(URI space): void	Unsubscribe
unsubscribe(URI space, Query query): void	Unsubscribe for the specific query

5. Conclusion

In this deliverable, we have presented the architecture of Semantic Spaces, a distributed and scalable semantics-driven data management infrastructure that will be used in the SOA4All Distributed Service Bus (DSB). The spaces become repositories for service descriptions and other data used by the various components developed in the project, and they are also the medium used for communication and coordination between multiple nodes of the DSB.

Semantic Spaces distribute the data of a space and the many spaces themselves through a combination of appropriate peer-to-peer overlay technologies. Spaces are distributed in a Chord ring, while the data is distributed in a CAN three-dimensional network (effectively a cube). We have summarized the principles behind these overlay technologies, and we have shown how exactly they can be applied to the distribution of spaces and data.

While spaces are fixed data containers that are put in place by administrators, clients can also set up *federations*, temporary scopes for data management. Federations can be set up either explicitly or indirectly by expressing relationships between spaces. A common kind of relationship is the subspace hierarchy, where a space may contain subspaces which themselves may contain further subspaces. A subspace hierarchy facilitates combining data from multiple spaces in a clear and intuitive way. Beside subspace hierarchies, spaces can be in other relationships, such as “similar spaces” and “affine spaces”, which may guide the clients in specific settings. Space relationship management increases the flexibility and manageability of Semantic Spaces.

The Semantic Space infrastructure of SOA4All exposes a simple set of operations for publishing and removal of RDF data, for executing queries (especially via SPARQL), and for subscribing to publication events. A further set of management operations support creation and deletion of spaces and federations.

Semantic Spaces in SOA4All are built on the conceptual results of the TripCom project, which has achieved significant and mature results at the level of specifications, also with respect to scalability and usability investigations in the context of service architectures. In SOA4All, we will investigate further distribution of spaces, and the use of spaces for service coordination.

This deliverable presents the architecture and design of the Semantic Spaces infrastructure, which will be implemented as the next step of this task. The implementation will become the Deliverable D1.3.2B. Afterwards, we will enhance our Semantic Spaces in the continuation of Task 1.3, leading to Deliverables D1.3.3A and D1.3.3B due in the second half of the project.

References

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

[2] B. Gladman: Risks to Safety and Security. Presentation at Scrambling for Safety 8, 2008.

[3] P.Th. Eugster, P.A. Felber, R. Guerraoui, and A.-N. Kermarrec: The many faces of publish/subscribe. ACM Computing Surveys 35(2), 2003: 114-131.

[4] M. Fitting: First-order logic and automated theorem proving. Springer, 1990.

[5] A. Fränkel, Y. Bar-Hillel, and A. Levy: Foundations of Set Theory. North Holland, 1973.

[6] G. Hohpe and B. Woolf: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Longman, 2003.

[7] R. Krummenacher, E. Simperl, and D. Fensel: Towards Scalable Information Spaces. In Workshop on New forms of reasoning for the Semantic Web: scalable, tolerant and dynamic (ISWC), 2007.

[8] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim: A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys Tutorials 7(2), 2005: 72-93.

[9] F. Manola and E. Miller: RDF Primer, W3C Recommendation, 2004.

[10] E. Prud'hommeaux and A. Seaborne: SPARQL Query Language for RDF, W3C Recommendation, 2008.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker: A scalable content-addressable network. In ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, 2001: 161-172.

[12] N. Shalom: The Scalability Revolution: From Dead End to Open Road - An SBA Concept Paper, GigaSpaces Technologies, 2007.

[13] J. Spivey: The Z Notation: A Reference Manual. Prentice Hall, 1992.

[14] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transaction on Networking 11(1), 2003: 17-32.

[15] van Lamsweerde, A. (2000). Formal Specification: a Roadmap. In Int'l Conference on Software Engineering (Conference on The Future of Software Engineering), 2000: 147-159.

Annex A. Additional Sequence Diagram

The following sequence diagram depicts the role and relationship of the five kernel components in the case of retrieval from an implicit federation. Federations are explained in Section 3.2.

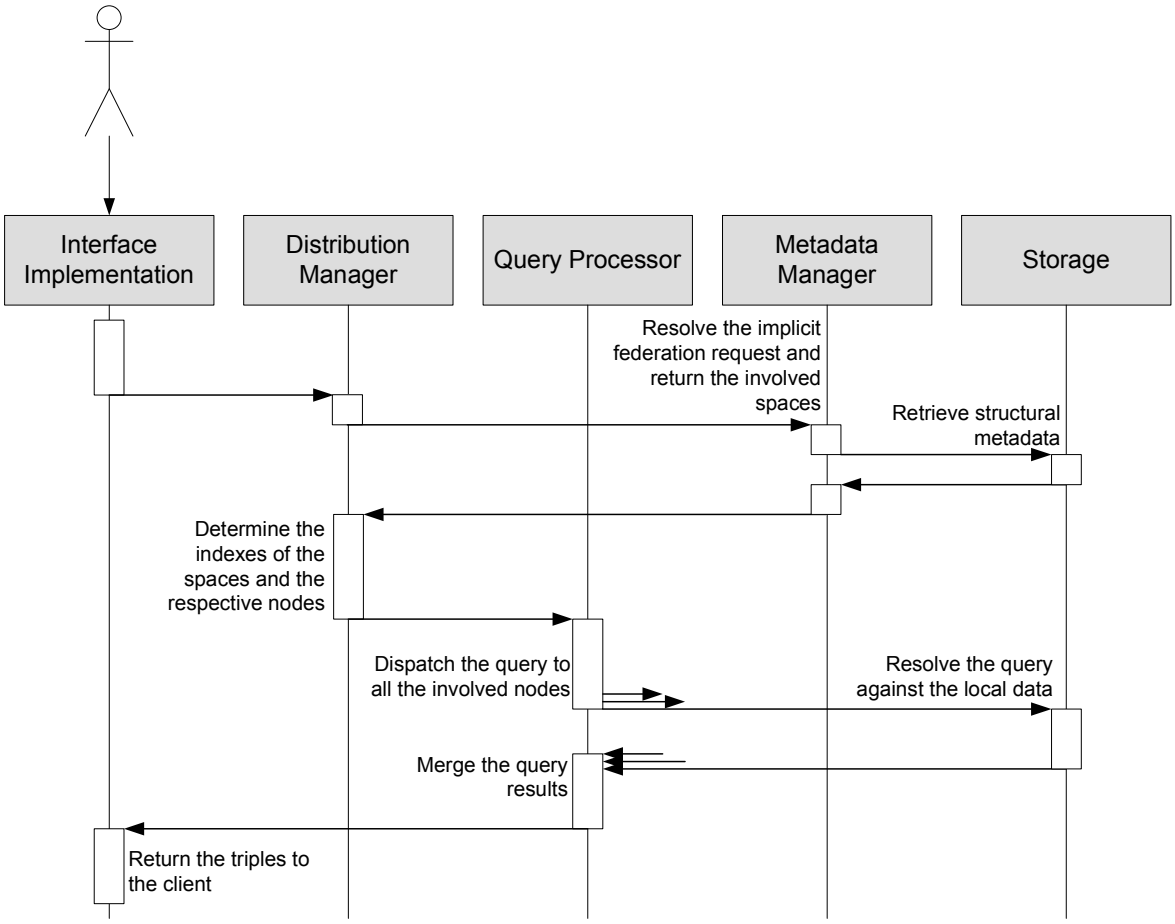

Figure 13: Sequence Diagram of the Retrieval From an Implicit Federation