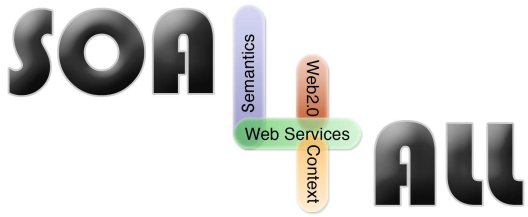

Project Number: **215219**

Project Acronym: **SOA4All**

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

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D1.3.3B Distributed Semantic Spaces: A Second Implementation

Activity N:	Activity 1
Work Package:	WP1
Due Date:	31/08/2010
Submission Date:	31/08/2010
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	UIBK
Revision:	1.0
Author(s):	Reto Krummenacher, Gerald Schrempf, Michael Fried (UIBK), Fabrice Huet, Laurent Pellegrino (INRIA)
Reviewers:	Guillermo Álvaro Rey (iSOCO), Mateusz Radzimski (ATOS)

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	2010-07-20	First TOC	Reto Krummenacher (UIBK)
0.2	2010-07-29	Semantic Spaces 2.0 Section	Gerald Schrempf (UIBK)
0.3	2010-08-04	Introduction/RDF2Go argumentation	Reto Krummenacher
0.4	2010-08-04	Configuration and Installation draft	Gerald Schrempf
0.5	2010-08-05	Distributed Space/P2P Overlay draft	Fabrice Huet (INRIA)
0.6	2010-08-06	Update to Distributed Space parts	Fabrice Huet
0.7	2010-08-10	Finalization Internal Draft	All
0.8	2010-08-16	Incorporation Review Feedback 1	All, Mateusz Radzimski (ATOS)
0.9	2010-08-18	Incorporation Review Feedback 2	All, Guillermo Alvaro Rey (iSOCO)
1.0	2010-08-19	Final release	Reto Krummenacher

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms

Acronym	Definition
API	Application Programming Interface
CAN	Content Addressable Network
GCM	Grid Component Model
GCMA	Grid Component Model Application
GCMD	Grid Component Model Descriptor
LOD	Linked Open Data
P2P	Peer-To-Peer
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
SVN	Apache Subversion
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

Executive summary

This deliverable describes the second and final implementation of the distributed semantic spaces infrastructure. The implementation provides the final realization of the concepts and specifications that were released with deliverable “D1.3.3A A Distributed Semantic Marketplace” [1]. The second implementation focused on offering a distributed semantic data management infrastructure that serves at once as virtualization layer for different RDF repositories and basic data management infrastructure and as RDF repository itself. For this reason, the implementation exposes semantic spaces as RDF2Go models that mimic and wrap various storage and RDF manipulation models such as the ones from OWLIM, Sesame or the P2P overlay developed within the SOA4All project. To this end, the semantic spaces core implementation including the space platform and the spaces models, as well as the overlay implementation are subject to this deliverable.

1. Introduction

This final software report about the distributed semantic spaces infrastructure describes the public Semantic Space 2.0 release that is available as open-source project via sourceforge.net and INRIA's GForge server, respectively, and as maven project via the public maven repositories for a facilitated integration and deployment. Moreover we are happy to note that the work is listed as implementation of the RDF2Go API on rdf2go.semweb4j.org. A priori the software release is the implementation of the conceptual updates to semantic spaces that were presented in deliverable D1.3.3A [2]. With respect to the alignment with the RDF models and operations provided through RDF2Go, the final implementation went a step further as initially described in D1.3.3A. The conceptual ideas presented at month M24 were considering making the Semantic Space API compatible to RDF2Go. The actual release presented in this document, however, implements the interfaces and makes semantic spaces particular types of RDF models. Further details about the reasons for this change and how it is realized is subject to Section 2. In short, thanks to this much closer alignment, the integration of the P2P overlay with RDF repositories and the use of semantic spaces as repositories are now much easier.

1.1 Purpose and Scope

This deliverable accompanies the software release of Semantic Spaces 2.0. The software release includes the semantic space logics, the RDF2Go models for spaces, the wrappers for various RDF repositories and data management endpoints, and an updated and evaluated implementation of the distributed semantic spaces layer, the P2P overlay. Although, as stated above, there are some differences between the implementation and the specification in deliverable D1.3.3A, the reader is still mostly referred to previous deliverables about semantic spaces for conceptual argumentations. This deliverable focuses only on an implementation overview, the updated API, installation and configuration information and some evaluation of the Semantic Space 2.0 release of SOA4All.

1.2 Structure of the document

In a first part, the deliverable presents the Semantic Space 2.0 software release in Section 2. There are two parts to the presentation: first, the Semantic Space Core implementation and APIs in Section 2.1, and second, in Section 2.2 a description of the P2P Overlay that enables distributed semantic spaces. Section 3 is dedicated to more information about the installation and configuration of different semantic space realizations; while the default release of Semantic Space 2.0 comes with a local Sesame store and a binding to public SPARQL endpoints, it is possible to use spaces based on OWLIM or the overlay upon request. Finally, with Section 4 we conclude the deliverable.

2. Software Description

The aim of Semantic Space 2.0 was, as described in deliverable D1.3.3A [1], the offering of more standardized RDF/SPARQL interfaces for the access to semantic spaces. While the role of semantic spaces as shared data management and virtualization infrastructure still remains the same, the interfaces were changed from mostly tuplespace-based to more RDF repository-aware. The updated architecture plan has foreseen to use of the RDF2Go framework as baseline for the interface specification. The semantic space operations, for example for publishing RDF, were consequently changed from write(URI space, Statement triple) to addStatement(URI space, Statement triple). The intention was to have the semantic space API mimic the RDF2Go API.

While this idea facilitated the understanding of semantic spaces and now supported (all) standard RDF manipulations such as adding knowledge, querying and removing data, the new API still caused unnecessary difficulties when moving from a standard RDF repository (e.g., Sesame or OWLIM) to semantic spaces within existing implementations. There was a need to change the signatures of the methods applied.

Indeed, the big advantage of the RDF2Go framework is the abstraction layer over many different repository implementations via the concept Model. The idea that is now followed with this final release of Semantic Spaces 2.0 is to expose spaces as models, according to RDF2Go, too. Accessing a semantic space is now equivalent to the manipulation of a model instance, which makes the switching from an OWLIM repository to the more expressive semantic space virtualization layer as simple as changing the model type – with no need for updates to the operations and operation signatures in an existing implementation.

Consequently, the operation to add statements no longer requires the space identifier as parameter, but the space is explicitly given as the object on which the operation is executed: mySpace.addStatement(Statement triple).

The remainder of this section describes the semantic spaces software package. In order to provide a well-structured approach to the software, we divide the overall software package into three parts:

- API specification and semantic spaces core implementation,
- Space models and RDF2Go extensions,
- Peer-to-peer distribution and indexing infrastructure.

2.1 Semantic Spaces 2.0

The Semantic Spaces 2.0 conceptual architecture (Figure 1, details to all parts are given in subsequent sections of this report) shows the API at the top-level with the two interfaces ISemanticSpace and ISpaceModel. The ISemanticSpace interface describes space-management-operations. The ISpaceModel interface describes the possible operations on a single space, and extends the RDF2Go Model interface accordingly.

The semantic spaces core implements the ISemanticSpace interface and holds references to the underlying spaces (models). The SpaceModel layer encapsulates all the individual spaces that are represented in a separate SpaceModel, each implementing the ISpaceModel interface.

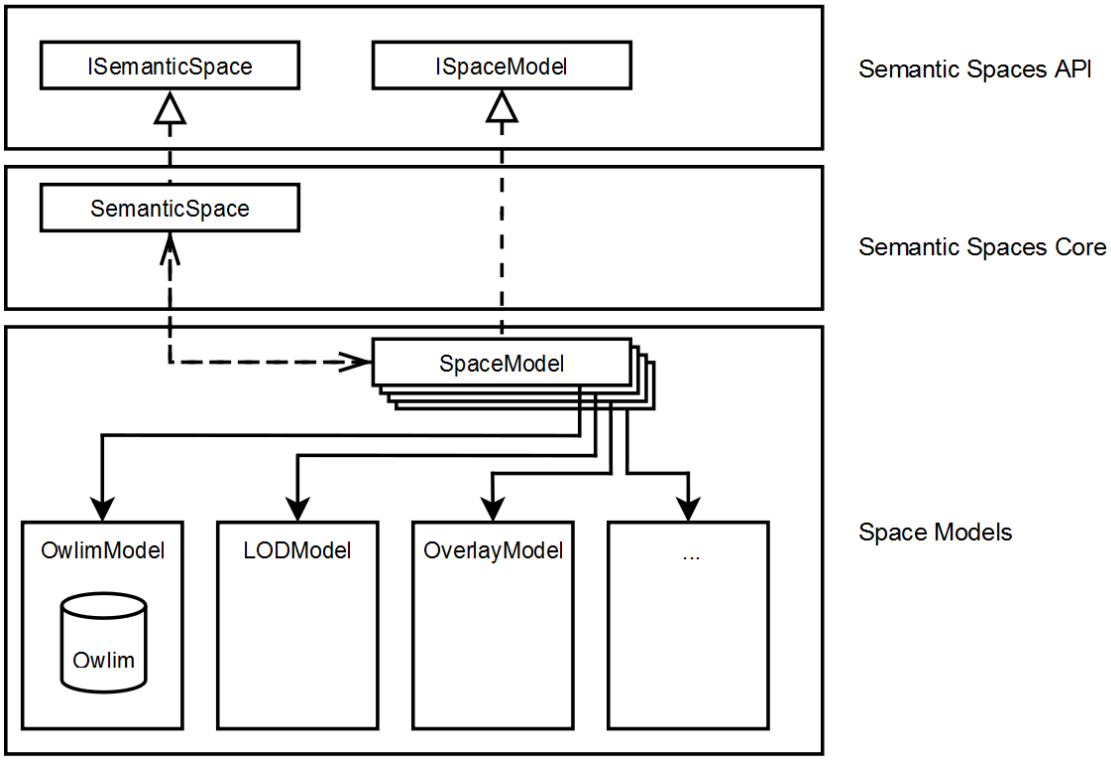

Figure 1: Architecture of Semantic Spaces

In comparison to Section 3.1 of D1.3.3A [1], the current architecture has been decomposed into a core implementation of the space infrastructure and the SpaceModel layer, so that, as described earlier in this section, the spaces can be maintained as separate models and are fully compatible to the RDF2Go model interface.[1] The semantic space core stores handles to space models and matches specific space identifiers (URIs) to SpaceModel; i.e., the core implementation can be queried for SpaceModel instances. The operations for publishing and retrieving semantic data such as addStatement or sparqlConstruct are then called without an additional space parameter directly on the corresponding SpaceModel instance.

The final SOA4All release of Semantic Spaces 2.0 currently allows the usage of different local repositories such as OWLIM and Sesame that are wrapped as space models, or distributed spaces that leverage the P2P overlays developed within the project (Section 2.2). Additionally, semantic spaces wrap public SPARQL endpoints as so-called LODModels in order to query linked data sources as spaces. A LODModel takes the URL of a public SPARQL endpoint and wraps the endpoint as semantic space. SPARQL queries are then forwarded to the URL, where they are resolved. Further information about the wrapping of SPARQL endpoints as spaces in LODModels can be found in Section 2.1.3.

2.1.1 Semantic Spaces 2.0 API

The Semantic Spaces API layer is discussed more concretely in this section; a component map is depicted on Figure 2 and Figure 3.

[1] http://rdf2go.semweb4j.org

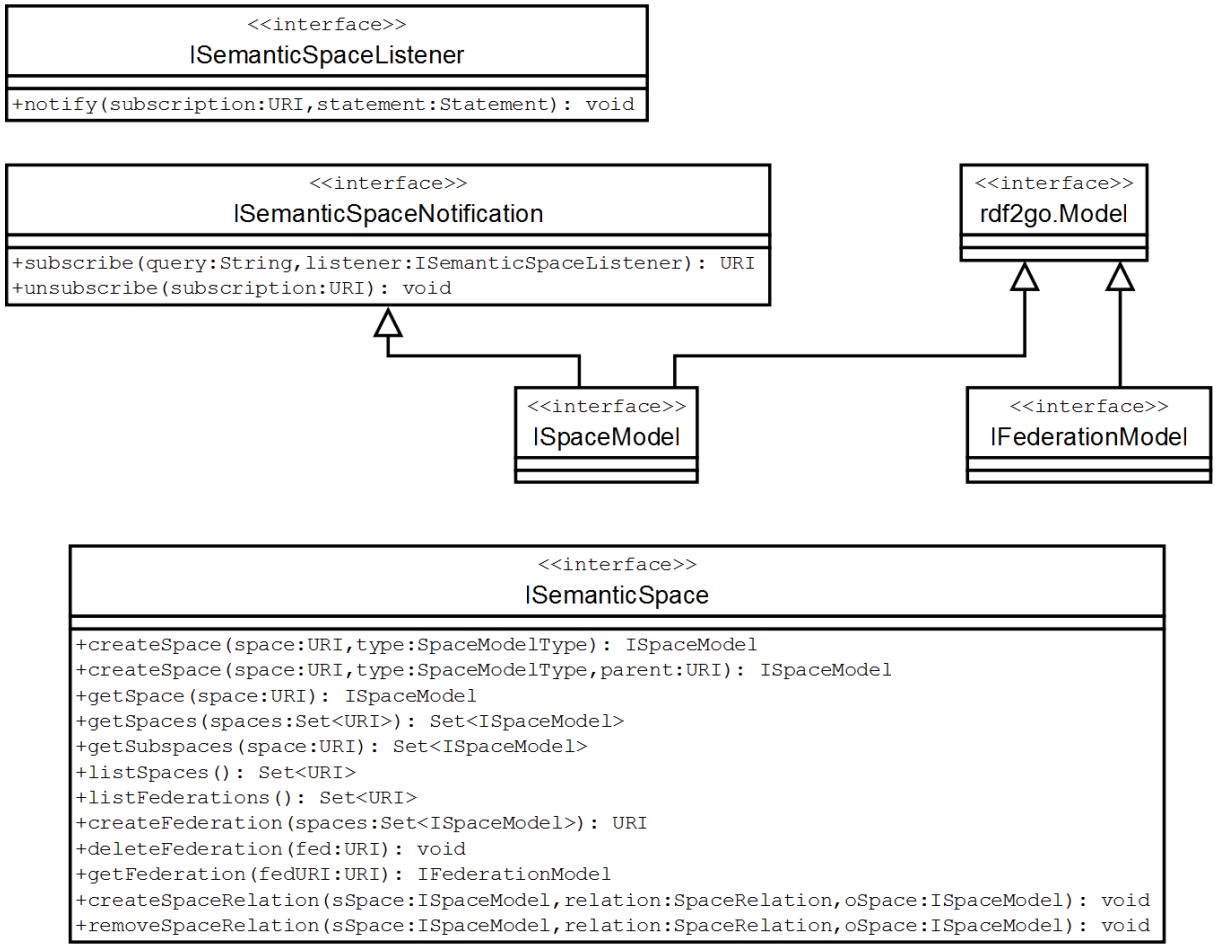

Figure 2: Semantic Spaces 2.0 API

The ISemanticSpace interface shows the public accessible methods of a Semantic Spaces Core. When creating a new space, the enumeration SpaceModelType defines the type of the space. Possible values are LocalRepository, DistributedSpace and Linked Open Data (LOD). The enumeration SpaceRelation is used to create or remove relations between spaces and has the possible values hasSubspace, isSubspaceOf, isSimilarTo, isRelatedTo and seeAlsoSpace. These space relations can also be found in Section 3.2 of Deliverable D1.3.2A [1].

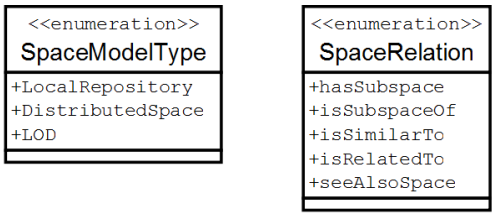

Figure 3: Semantic Spaces 2.0 Enumerations

The ISpaceModel interface extends the RDF2Go Model interface and the ISemanticSpaceNotification. The ISemanticSpaceNotification interface is used in combination with the ISemanticSpaceListener interface for enabling the notification-mechanism on spaces. Further information for subscriptions can be found in Section 3.5 of Deliverable D1.3.3A [1].

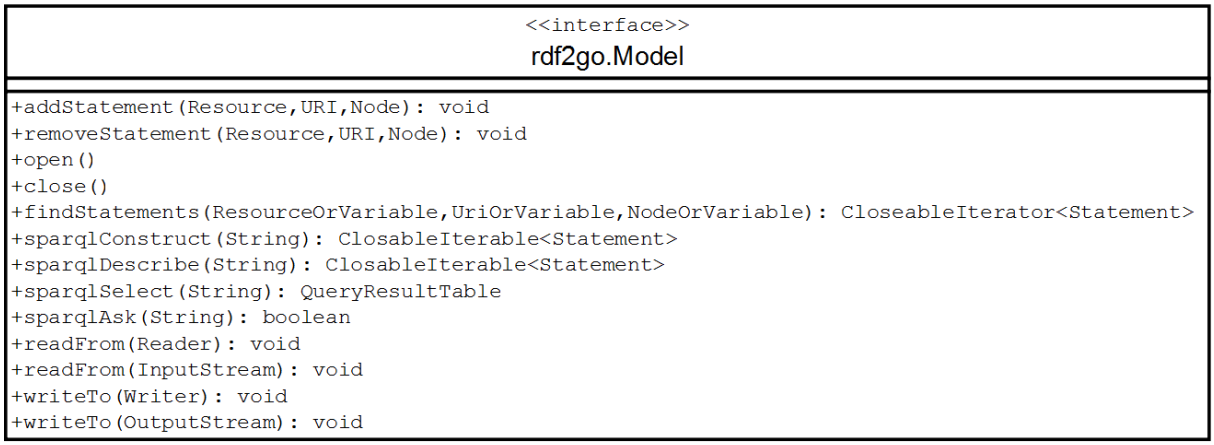

Figure 4: Important operations of the RDF2Go Model interface

Figure 4 shows the most important operations of the RDF2Go Model interface. There are also some convenience methods specified by the RDF2Go Model interface for operations like addStatement, removeStatement, readFrom and writeTo with different parameters which can be seen in the RDF2Go online javadoc.[2]

2.1.2 Semantic Spaces Core

The Semantic Spaces Core implements the ISemanticSpace interface (Figure 2), which provides operations to create semantic spaces, get spaces by their identifying URI, get the subspaces of a specific space, list all spaces, create federations of a set of spaces, delete such federations and create relations like “hasSubspace” and “isSubspaceOf” between spaces, as initially defined in D1.3.2A [1] and D1.3.3A [1].

The core manages a metadata-space to store information about relations between spaces like the subspace hierarchy or the underlying storage-type of a specific space. It also stores federations and holds references to a local repository and an overlay (optional).

Every underlying storage part like OWLIM, Sesame or the Overlay has to provide a model factory, which implements the RDF2Go ModelFactory interface. This factory is loaded by the “RDF2Go.register” method and then used to create new spaces (models). The type of a space is defined by the used model factory. This provides better flexibility of the used storage parts, because the dependencies are checked at runtime and do not have to be available at compile time if they are not needed.

To use specific settings for the Semantic Spaces, properties or the filename of a properties-file can be passed when creating a new Semantic Spaces Core. Possible settings are the storage-path of the local repository, the type of the local repository via its ModelFactory (e.g., BigOWLIM, Sesame) and the usage of an overlay. Further information about the properties and how to configure those can be found in Section 3.

2.1.3 SpaceModel: Wrapping RepositoryModels

Every different underlying model that implements the RDF2Go Model interface can be wrapped by a SpaceModel. The SpaceModel also implements the RDF2Go Model interface

[2] http://mavenrepo.fzi.de/semweb4j.org/site/rdf2go.api/apidocs/org/ontoware/rdf2go/model/Model.html

and extends the RDF2Go AbstractModel class. On the one hand, the semantic spaces model virtualizes the underlying repository models – possibly a public SPARQL endpoint too via LODModel – and, more importantly, adds an additional abstraction layer that is used to implement various semantic space specific features and functionalities like executing SPARQL queries or findStatement methods recursively also on subspaces or across federations.

As the underlying models all implement the RDF2Go Model interface, recursive calls to spaces and their subspaces are independent of their type.

The notification mechanism that is exposed by the SpaceModel class is currently only applicable if the corresponding space is implemented on top of a BigOWLIM repository. No other currently available storage infrastructure allows for subscriptions to RDF patterns. Further information about the SOA4All notification services that are now shipped with BigOWLIM can be found in Section 3.5 of deliverable D1.3.3A.

2.1.4 Exceptions

There are different exceptions in the Semantic Spaces 2.0 API:

- An OperationNotSupportedException is thrown if operations are called which are not possible. Examples are the not supported subscription mechanism on a Sesame repository or the addStatement methods when working with a LOD model.
- The SpaceAlreadyExistsException is thrown if a space URI already exists.
- The SpaceNotExistsException is thrown if a space does not exist.
- The SpaceTypeNotSupportedException is thrown if a space with a not allowed type should be created, e.g. if the distributed spaces part is disabled in the properties when creating a Semantic Spaces Core.
- The NoSparqlEndpointException is thrown if a given space URI does not map to a valid SPARQL endpoint when trying to create a LODModel space.

Figure 5 shows the exceptions hierarchy of the presented Semantic Spaces 2.0 release:

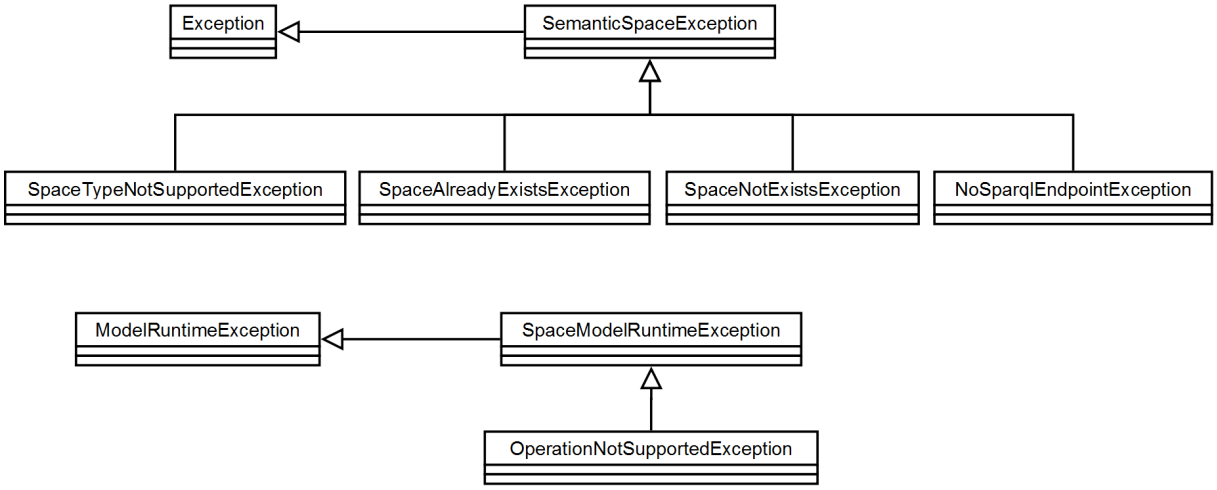

Figure 5: Exceptions of the Semantic Spaces 2.0 API

2.2 Distributed Spaces and P2P Overlay

The distributed space implementation has been re-factored to implement the new Semantic Spaces 2.0 API and improve the overall performance. The overlay implementation leverages the RDF2Go API and is exposed to the Semantic Space Core as RDF2Go model. Internally, it can now support any storage framework with an RDF2Go interface. A global description of the architecture can be found in D1.3.3A [2], and in the sections above. This part of the report focuses on the distributed implementation, depicted in Figure 1 as OverlayModel.

2.2.1 Semantic Space modular architecture

One of the goals when designing this distributed storage was to be able to easily change or modify some parts. A modular architecture is at the heart of the design, clearly separating the infrastructure (a CAN overlay), the query engine (e.g., OWLIM, Sesame) and the storage system (a BigOWLIM repository). However, these elements do not work in isolation, rather they require frequent interactions. In this section, we will outline the different parts of our architecture, explaining their functions and showing their relations.

2.2.1.1 Peer Architecture

A peer is the entity responsible for maintaining the CAN infrastructure, routing messages and accessing the local repository.

The 3D CAN overlay is managed through an Overlay object which is responsible for maintaining a description of the zone managed by the current peer and an up-to-date list of neighbours. Changing the number of dimensions of the CAN, e.g., to handle meta-data, requires providing a modified implementation of the Overlay object.

To route a query, it is first analyzed in order to determine the constant parts, if any, which will be used to direct it to the target peer. When there is not enough information to make a decision, it is broadcasted to the neighbouring peers, which will perform the same process.

2.2.1.2 Query Analysis and Manipulation

Although the routing of the query is a peer's responsibility, part of the process requires the query analysis to extract atomic queries and their constant parts. We have delegated this part to specific libraries, which offer dedicated operations for SPARQL query manipulation. When a query returns data sets from multiple peers, the merge/join operation is also delegated. In order to experiment with the modularity aspect of our implementation, we have used both Sesame and Jena without impacting the other parts of the architecture.

2.2.1.3 Storage Abstraction

The storage is ultimately responsible for storing data and locally processing queries. It is important for the peer-to-peer infrastructure to be independent from the storage implementation. All references are isolated through an RDF2Go abstraction layer whose role is to manage the differences between data-structures and API between the peer-to-peer and the storage implementations. Some requests require accessing the local repository to read or write some information. Although this is rather straightforward, some care has to be taken regarding the commit of data to the storage. With some implementations like BigOWLIM, committing can take some time and thus should not be done after each write operation. The

peer can implement a policy to only perform them when a threshold is reached (e.g., time since last commit aka *commit interval*, number of write done, etc.) or when a read query has to be processed.

The overall interaction between the subcomponents of the architecture can be seen in Figure 6. A request is transmitted through *Peer 1*, which performs some analysis, and propagates it through various overlays object, before finally reaching the target local storage.

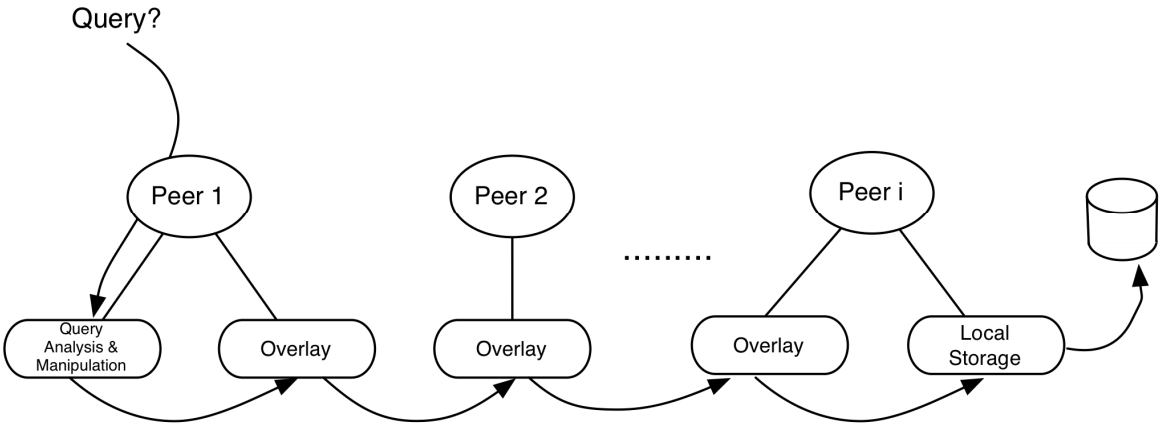

Figure 6 : Interactions from the query initiator to the destination peer

2.2.2 Performance improvement

A lot of work has been dedicated to improving the performance of the implementation and to significantly reduce the time to perform operations such as adding statements to the space or performing queries.

The following is a list of some of the optimisations performed:

- Refactoring of messages: now, responses do not contain the initial query but only the required information to route back the matching statements. For example, the size of messages to insert a statement in the network has been reduced by 80%, from 6.5KB to 1.3KB.
- Responses associated to messages are no more merged at each synchronization point but only when the response is received by the initiator of the request. Moreover, data retrieved for response are serialized once and stored as byte array in order to avoid multiple serializations.
- All the classes generated by the ProActive middleware [6] at runtime are created at compile time with a custom Maven plugin (open source project available at http://code.google.com/p/proactive-maven-plugin/).
- Querying the data store is done in parallel to the forwarding of the request to the neighbours, to overlap computation and communication.

Extensive experiments have been conducted and results are presented in D1.5.3.

3. Installation and Configuration

3.1 Installation

3.1.1 Semantic Spaces 2.0 Core

The Semantic Spaces are released as maven project and deployed on the STI Innsbruck maven repository. It can be included by adding the following repositories to a maven pom file:

```
<repositories>
  <repository>
    <id>sti2-archiva-external</id>
    <url>http://maven.sti2.at/archiva/repository/external</url>
  </repository>
  <repository>
    <id>sti2-archiva-snapshots</id>
    <url>http://maven.sti2.at/archiva/repository/snapshots</url>
  </repository>
</repositories>
```

The current release version is 0.0.2, the current development version is 0.0.3-SNAPSHOT.

Adding the semanticspaces-impl artifact to the maven dependencies provides the semantic spaces entry point implementation (the core):

```
<dependencies>
  <dependency>
    <groupId>eu.soa4all.semanticspaces</groupId>
    <artifactId>semanticspaces-impl</artifactId>
    <version>0.0.2</version>
  </dependency>
</dependencies>
```

The latest development releases can be used by adding the unstable SNAPSHOT version:

```
<dependencies>
  <dependency>
    <groupId>eu.soa4all.semanticspaces</groupId>
    <artifactId>semanticspaces-impl</artifactId>
    <version>0.0.3-SNAPSHOT</version>
  </dependency>
</dependencies>
```

To use the implementation together with a local OWLIM repository the artifact owlim-rdf2go has to be included as well - it offers an RDF2Go adapter for OWLIM. To use the implementation together with the SOA4All P2P Overlay for distributed spaces, the artifact semanticspaces-overlay has to be included. The source code of the working version 0.0.2 including the OWLIM RDF2Go adapters is now also available publicly on sourceforge.net:

http://semanticspaces.svn.sourceforge.net/viewvc/semanticspaces/trunk/

The Semantic Spaces API consists of the following packages and classes:

1. at.sti2.semanticspaces.api
    - IFederationModel, ISemanticSpace, ISemanticSpaceListener, ISemanticSpaceNotification, ISpaceModel
2. at.sti2.semanticspaces.api.enums
    - SpaceModelType, SpaceRelation
3. at.sti2.semanticspaces.api.exceptions
    - OperationNotSupportedException, NoSparqlEndpointException, SemanticSpaceException, SpaceAlreadyExistsException, SpaceModelRuntimeException, SpaceNotExistsException, SpaceTypeNotSupportedException
4. at.sti2.semanticspaces.api.spaceOntology
    - SpaceOntology

The core implementation of the semantic space platform consists of the following packages and classes:

1. at.sti2.semanticspaces.impl
    - PropertyReader, SemanticSpace
2. at.sti2.semanticspaces.impl.model
    - FederationModel, LODModel, ModelHelper, SpaceModel
3. at.sti2.semanticspaces.impl.queries
    - AskQuery, ConstructQuery, DescribeQuery, SelectQuery, SparqlQuery

3.1.2 Distributed Spaces and P2P Overlay

The source code has been moved from the previously private SVN server of the SOA4All project to a new public one hosted by INRIA. The public SVN is available through anonymous access:

https://gforge.inria.fr/scm/viewvc.php/?root=dspace

The following modules compose the source tree:

- proactive-structuredp2p: CAN and Chord implementation, based on the ProActive middleware.
- dspace-overlay: overlay implementation for semantic data.
- dspace-overlay-performance: simple tests used by the continuous integration server to measure the performance of the overlay implementation on a local machine when a commit is performed.
- dspace-overlay-benchmarks: full scale experiments based on random data or a subset of the Berlin SPARQL Benchmark [5].
- dspace-overlay-samples: scripts and utility classes for deploying a distributed semantic space.
- dspace-rdf2go: OverlayModel and OverlayModelFactory compatible RDF2Go.

The compilation process is managed by Maven and each module has a *pom.xml* file which specify dependencies. Running the *mvn install* command from the trunk directory will compile all modules, generate the required classes and perform the unit tests.

To make the deployment process easier, the distributed spaces artifacts are released as maven project and deployed on the dspace gforge maven repository. Artifacts can be included in another maven projects by adding the following section to the maven pom file wished:

```
<repositories>
  <repository>
    <id>dspace-releases</id>
    <url>http://dspace.gforge.inria.fr/maven2/repository/releases</url>
  </repository>
  <repository>
    <id>dspace-snapshots</id>
    <url>http://dspace.gforge.inria.fr/maven2/repository/snapshots</url>
  </repository>
</repositories>
```

The current release version is 0.0.1, the current development version is 0.0.2-SNAPSHOT.

To have access to the OverlayModel and OverlayModelFactory classes, only the fr.inria.gforge.dspace:dspace-rdf2go artifact has to be added to the maven dependencies section. All others dependencies are included by transitivity.

```
<dependencies>
  <dependency>
    <groupId>fr.inria.gforge.dspace</groupId>
    <artifactId>dspace-rdf2go</artifactId>
    <version>0.0.1</version>
  </dependency>
</dependencies>
```

3.2 Configuration

3.2.1 Semantic Spaces 2.0 Core

To configure the Semantic Space implementation, a properties object or a path string can be passed to the constructor of the Semantic Spaces Core.

The configuration file allows the following properties:

Property	Possible values
Storage	path for storage of local repository
Reasoning	“OWL”
repository	“org.openrdf.rdf2go.RepositoryModelFactory” “com.ontotext.trree.rdf2go.OwlimModelFactory”
overlay (optional)	“eu.soa4all.dsb.space.overlay.OverlayModelFactory”

An example properties file can be seen in Table 1.

Table 1: Example property file

```
###
### Configuration-File for using BigOWLIM or Sesame as Repository.
### Author: Gerald Schrempf, gerald.schrempf@sti2.at
### STI Innsbruck, 2010
###

### Storage-parameter can be variated by the user.
Storage = /tmp/space-repository

### Reasoning has to be supported by the used RDF2GO-ModelFactory.
Reasoning = OWL

### Choose between BigOWLIM or Sesame. (only one repository allowed)
### Make sure that the needed JAR-Files are locally available.
repository = org.openrdf.rdf2go.RepositoryModelFactory
#repository = com.ontotext.trree.rdf2go.OwlimModelFactory

### Enable distributed spaces (overlay).
### Make sure that the needed JAR-Files are locally available.

#overlay = eu.soa4all.dsb.space.overlay.OverlayModelFactory
```

A Semantic Spaces Core can be instantiated with:

```
SemanticSpace semSpaceImpl = new SemanticSpace("config.properties");
```

If no properties-parameter is passed to the core, the SemanticSpace class will use an internal default config with a local Sesame repository, the SPARQL endpoint functionality and no overlay. Some examples for the usage can be found in different test classes on sourceforge.net in the semanticspaces-integration-test module.

The default configuration of the Semantic Spaces 2.0 release has a fixed dependency on Sesame. This allows the default usage of a local Sesame repository and the binding of SPARQL endpoints to query online linked data sources.

3.2.2 Distributed Spaces and P2P Overlay

The distributed space implementation can use one or several BigOWLIM repositories as local storage. The location of the repositories can be changed on each machine by defining a property in a properties file.

Property	Default Value
repositories.path	$INSTALLATION_DIR/owlim/repositories

By default, the property is searched in the $HOME_DIR/.dspace/space.properties file. The location of the properties file can be changed by running the distributed space application with the '-Ddspace.configuration=path/to/properties/file' Java property.

All the necessary scripts for running the distributed space are located in the dspace-overlay-samples module. The infrastructure (list of machines, path to JVM…) is described in specific XML files called GCM deployment files:

- GCMA.xml: contains all the requirements of the application, such as the path to configuration files, to the Java Virtual Machine and a reference to node providers.
- GCMD.xml: describe the nodes providers for the application, i.e. the mechanisms used to acquire machine where to run the semantic space peers.

An example of a GCMA.xml file is given bellow. It indicates that the application will be given a virtual node called *trackersVN* which will correspond to hosts described in the file GCMD-Trackers.xml.

```
<environment>
   <javaPropertyVariable name="dspace.bundle.home" />
   <javaPropertyVariable name="java.home" />
   <javaPropertyVariable name="user.home" />
</environment>
<application>
   <proactive base="root" relpath="${dspace.bundle.home}">
      <virtualNode id="trackersVN" capacity="1">
         <nodeProvider refid="trackersNP" />
      </virtualNode>
   </proactive>
</application>
<resources>
   <nodeProvider id="trackersNP">
      <file path="GCMD-Trackers.xml" />
   </nodeProvider>
</resources>
```

The GCMD-Trackers.xml indicates that the resources we are looking for will be created by doing an ssh (*sshLan*) to the machine eon1.inria.fr.

```
<infrastructure>
   <hosts>
      <host id="ComputeNode" os="unix" hostCapacity="1" vmCapacity="1">
         <homeDirectory base="root" relpath="${user.home}" />
      </host>
   </hosts>
   <groups>
      <sshGroup id="sshLan" hostList="eon1.inria.fr" />
   </groups>
</infrastructure>
```

It is beyond the scope of this deliverable to fully describe the ProActive deployment mechanism. Interested readers should refer to [7].

The distributed semantic space distribution comes with a script which can be used to easily deploy a semantic space on a local machine or on a cluster (i.e. a set of machines accessible through an ssh connexion): *benchmarks-launcher.sh.*

Some examples of preconfigured GCMA and GCMD files are given in the deployment directory from dspace-overlay-samples module:

GCMD-CAN-Peers.xml GCMD-Chord-Peers.xml GCMD-NodeProvider.xml GCMD-Trackers.xml	Default GCM files for deploying a space on a cluster using ssh connexions.
GCMA-Local.xml GCMD-Local.xml	Default GCM files for deploying a space on the local machine. This is useful for testing and debugging.

4. Conclusions

This report was an adjunct to the final software release of the SOA4All distributed semantic spaces infrastructure. The deliverable described the Semantic Spaces 2.0 open-source project. The report explains where to access the code and binaries, and how to install and configure Semantic Spaces 2.0 with different application settings in mind.

Although conformant to the specification published with D1.3.3A [2], the presented release reflects some further design decisions that were taken in order to improve the applicability and usability of the SOA4All semantic spaces.[3] The spaces now fully mimic repository models as they are defined by the RDF2Go abstraction framework. To this end, the integration of the P2P overlay with RDF repositories and the use of semantic spaces as repositories are now much easier and faster. Moreover, this much closer alignment with RDF2Go resulted in Semantic Spaces 2.0, and in particular the OWLIM adapter, being listed as official RDF2Go implementations at rdf2go.semweb4j.org.

[3] The added simplicity of adoption was effectively proven by an updated implementation of the service composition example over semantic spaces as presented in [1] that is being done for the presentation and demonstration of Linked Open Services at FIS2010 [3],[4].

5. References

1. Reto Krummenacher, Imen Filali, Fabrice Huet and Francoise Baude: Distributed Semantic Spaces: A Scalable Approach To Coordination, SOA4All Project Deliverable D1.3.2A v1.1, August 2009.
2. Reto Krummenacher, Fabrice Huet, Michael Fried, Laurent Pellegrino, Ivan Peikov, and Alex Simov: A Distributed Semantic Marketplace. SOA4All project deliverable D1.3.3A, March 2010.
3. Reto Krummenacher, Barry Norton, and Adrian Marte: Towards Linked Open Services and Processes. 3rd Future Internet Symposium, September 2010.
4. Reto Krummenacher, Barry Norton, and Adrian Marte: Linked Open Services: Update on Implementations and Approaches to Service Composition. 3rd Future Internet Symposium (Poster & Demo), September 2010.
5. Chris Bizer und Andreas Schultz: Berlin SPARQL Benchmark (BSBM) Specification – V2.0, December 2008 at http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/spec/.
6. ActiveEon, INRIA : ProActive, http://proactive.inria.fr
7. ActiveEon, INRIA : The ProActive GCM Deployment, http://proactive.inria.fr/4.3/Programming/ReferenceManual/multiple_html/GCMDeployment.html