

Project Number: **215219**  
 Project Acronym: **SOA4All**  
 Project Title: **Service Oriented Architectures for All**  
 Instrument: **Integrated Project**  
 Thematic Priority: **Information and Communication Technologies**

## D1.5.3 Testbeds Validation

<b>Activity N:</b>	1
<b>Work Package:</b>	1
<b>Due Date:</b>	30/09/2010
<b>Submission Date:</b>	01/10/2010
<b>Start Date of Project:</b>	01/03/2008
<b>Duration of Project:</b>	36 Months
<b>Organisation Responsible of Deliverable:</b>	HANIVAL
<b>Revision:</b>	1.0
<b>Author(s):</b>	Bernhard Schreder (Hanival), Juan Luis Prieto Martínez (ATOS), Matteo Villa (TXT), Giovanni Di Matteo (TXT), Claudio Stella (TXT), Fabrice Huet (INRIA), Elton Mathias (INRIA)
<b>Reviewers:</b>	Alex Simov (Ontotext)

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

## Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	15/07/10	ToC	Bernhard Schreder (Hanival)
0.2	30/07/10	Section 2 added	Bernhard Schreder (Hanival)
0.3	04/08/10	Updates to Section 2.2	Claudio Stella (TXT), Matteo Villa (TXT)
0.4	05/08/10	Updates to all sections	Bernhard Schreder (Hanival)
0.5	10/08/10	Section 3.2	Fabrice Huet (INRIA)
0.6	11/08/10	Updates to Section 2 and 3	Elton Mathias (INRIA)
0.7	27/09/10	Updates to Section 2 and 3	Juan Luis Prieto Martínez (ATOS)
0.8	29/09/10	Conclusion and final updates, version sent to reviewers	Bernhard Schreder (Hanival)
1.0	30/09/10	Final version	

# Table of Contents

<b>EXECUTIVE SUMMARY</b>	<b>6</b>
<b>1. INTRODUCTION</b>	<b>7</b>
1.1 PURPOSE AND SCOPE	7
1.2 STRUCTURE OF THE DOCUMENT	7
1.3 ALIGNMENT TO SOA4ALL EVALUATION	8
<b>2. SOA4ALL TESTBED INFRASTRUCTURE</b>	<b>9</b>
2.1 OVERVIEW OF THE TESTBED INFRASTRUCTURE: SERVICE PARKS	9
2.2 WEB SERVICE GENERATION	11
2.2.1 <i>Genesis</i>	11
2.2.2 <i>REST Services Support for Genesis</i>	11
2.2.3 <i>Technical Implementation</i>	12
2.2.4 <i>Installation</i>	16
2.2.5 <i>An Example</i>	16
<b>3. SOA4ALL RUNTIME EVALUATION</b>	<b>19</b>
3.1 EVALUATION SCENARIOS	19
3.1.1 <i>fDSB Evaluation</i>	19
3.1.2 <i>Cloud Bursting Scenario</i>	21
3.1.3 <i>Distributed Space</i>	23
3.2 RUNTIME EVALUATION RESULTS	24
3.2.1 <i>fDSB Evaluation</i>	24
3.2.2 <i>Semantic Spaces evaluation</i>	24
3.3. COMPARISON WITH OTHER SOLUTIONS	29
<b>4. CONCLUSIONS</b>	<b>31</b>
<b>5. REFERENCES</b>	<b>32</b>
<b>ANNEX A.</b>	<b>33</b>

## List of Figures

Figure 1: GENESIS Architecture .....	11
Figure 2: REST Service Generation with GENESIS .....	12
Figure 3: Genesis for REST logical architecture .....	13
Figure 4: Diagram of new classes for package at.ac.tuwien.vitalab.genesis.model .....	14
Figure 5: Execution of .bat files .....	18
Figure 6: Final deployment .....	18
Figure 7: Testbed deployment .....	20
Figure 8: fDSB Service Invocation Path.....	21
Figure 9 Cloud Bursting.....	22
Figure 10: Cloud Bursting deployment.....	23
Figure 11: Individual time for sequential insertion of random statements on a single local peer .....	25
Figure 12: Individual time for sequential insertion of random statements on a remote peer ...	25
Figure 13: Insertion of 1000 statements for variable number of peers, 1 thread (left) and 32 threads (right) .....	26
Figure 14: Evolution of the time for concurrent insertion on a 100 peers overlay .....	26
Figure 15: Custom queries with BSBM dataset on various overlays, execution time (left) and message overhead (right).....	29
Figure 16 Gateways Scenario .....	30

## List of Tables

Table 1: Experimental fDSB Deployment Resources.....	19
Table 2: fDSB Average Invocation Times .....	24

## Glossary of Acronyms

Acronym	Definition
API	Application Programming Interface
D	Deliverable
DSB	Distributed Service Bus
EC	European Commission
EPR	Endpoint Reference
ES	Enterprise Service
ESB	Enterprise Service Bus
EU	European Union
fDSB	Federated DSB
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
OSS	Operations Support System
REST	Representational State Transfer
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SUT	System under Test
URI	Uniform Resource Identifier
VM	Virtual Machine
WADL	Web Application Description Language
WAR	Web Application Archive
WP	Work Package
WS	Web Service
WSDL	Web Service Description Language

## Executive summary

Task 1.5 is concerned with the technical evaluation of the project, and its results can be used to validate the major technical objectives of SOA4All, including scalability and performance of the developed solutions. In this deliverable, we continue with the development and deployment of a testbed environment for SOA4All, which was first described in deliverable D1.5.1. This deliverable describes the final setup of the testbed environment and contains an evaluation of the results obtained through performing different sets of tests and comparing the results to alternative solutions. The deliverable is divided into two main sections.

The first part of the deliverable describes the overall testbed infrastructure, which enables testers and component owners to define configurable testbeds and services according to a collection of service templates, and consists of a diverse deployment of fDSB nodes over various domains.

The second major part of the deliverable defines the various evaluation scenarios used for performance testing of the WP1 results. Each evaluation scenario consists of a set of test cases which are performed on various testbeds. The results of these tests have been collected and are evaluated according to the metrics defined previously in deliverable D1.5.2. Related solutions for both the fDSB and the semantic spaces, which are the main technical results of WP1, are briefly described and available performance measurements are compared to the results obtained by the tests.

# 1. Introduction

This deliverable describes the continuation of the work in the scope of Task 1.5, the SOA4All Testbed infrastructure and evaluation of project results. According to the work done and described in deliverable D1.5.1 [5] and D1.5.2 [6], the testbed infrastructure has been developed. This deliverable now continues to describe the final set-up of the testbed environment.

In addition, the deliverable describes the different evaluation scenarios and test cases developed for the validation of the runtime environment, as well as the results of those tests. This also includes a comparison to other available solutions, in order to properly evaluate the results obtained by the performance experiments.

## 1.1 Purpose and Scope

As mentioned above, this deliverable describes the different activities to realise a testbed environment and is separated in two main sections.

The evaluation of the SOA4All runtime is based on the deployment and management of nodes of the Distributed Service Bus. The deliverable provides a detailed description of the set-up of the testbed infrastructure, based on the deployment of DSB nodes, in order to achieve the necessary scope to evaluate the scalability and performance of the SOA4All runtime.

The testbed infrastructure also enables testers and component owners to define configurable testbeds and services according to a collection of service templates, which are described in this deliverable and are aligned to the SOA4All Use Case storyboards (as detailed in [3], [7] and [2]).

The second major part describes the evaluation scenarios, specific test cases and other information for the actual evaluation of the runtime environment. The section collects the results of these tests and the evaluation of these results based on the metrics defined previously and comparable technical solutions.

## 1.2 Structure of the document

This document is structured as follows: following this introductory section, Section 2 of this document describes the overall testbed infrastructure, which enables testers and component owners to define configurable testbeds and services according to a collection of service templates, and consists of a diverse deployment of fDSB nodes over various domains.

Section 3 of the deliverable then defines the various evaluation scenarios used for performance testing of the WP1 results. Each evaluation scenario consists of a set of test cases which are performed on various testbeds. The results of these tests have been collected and are evaluated according to the metrics defined previously in deliverable D1.5.2. Related solutions for both the fDSB and the semantic spaces, which are the main technical results of WP1, are briefly described and available performance measurements are compared to the results obtained by the tests.

Finally, the deliverable concludes with a summary of the obtained results from the experiments and an outlook on additional ongoing experiments on the SOA4All testbed infrastructure. As, by the time of this writing, additional results are still collected, this deliverable will be updated by M33 of the project, in order to reflect the gained insight into the performance and scalability of the SOA4All runtime, specifically the fDSB and the Semantic Spaces.

### 1.3 Alignment to SOA4All Evaluation

The testbed infrastructure specified in this deliverable has been used to evaluate the main objectives of the project from a technical perspective. The main roadmap for evaluation was first summarised as part of deliverable D2.5.1 [4], and includes a set of metrics and performance indicators for the technical evaluation. Results from the evaluation process concerning these indicators are reported in this deliverable as well.



## 2. SOA4All Testbed Infrastructure

In order to demonstrate the distributed nature of the SOA4All infrastructure, the project established by month M18 a Distributed Service Bus implementation across three distinct nodes at three different locations. There are currently bus nodes, with co-located semantic space nodes, installed at eBM WebSourcing in Toulouse, France, at INRIA in Sophia Antipolis, France, and at the University of Innsbruck in Austria. While this is sufficient for a first implementation and to showcase the distributed nature of the SOA4All infrastructure, a three-node deployment is not considered well enough for evaluation and future uses. In particular, elements such as scalability and performance cannot adequately be measured, analysed and evaluated.

In this section, we therefore present the different testbeds that were used for a multi-level deployment plan for SOA4All that allows flexible scaling out in terms of machines that share the Distributed Service Bus. We first present the overall approach that is envisaged, and in a second subsection we present in more detail the various projects involved.

### 2.1 Overview of the Testbed Infrastructure: Service Parks

As presented in [10], one of the main goals of the fDSB is to offer a communication layer connecting service parks in a transparent way, despite of network configurations that might prevent direct connection of nodes hosting DSB nodes. Implementation details, installation and configuration are detailed in [11].

In this section, we present more details about the testbed used in the evaluation of the Federated DSB (fDSB). In order to assess the worthiness and performance of the fDSB, we carried out a series of experiments involving service parks deployed in different administrative domains, with different network configurations and access policies. This environment is composed by three service parks, each one deployed in a different administrative domain, including INRIA Sophia Antipolis, the Amazon EC2 cloud platform and Grid5000, the French experimental Grid infrastructure.

#### INRIA – Sophia Antipolis cluster

The INRIA private cluster used in the testbed is composed by 20 nodes with 1Gb Ethernet connectivity. Each node has 16GB of memory and two Intel E5335 processors, for a total of 8 cores on each node.

Because of INRIA network security, cluster nodes (and therefore the DSB which is running on these nodes) cannot be accessed by nodes outside of the secured INRIA network. The only available entrypoint is a gateway machine which only supports SSH connections. In spite of that, cluster nodes can access the external network (i.e. the Internet).

At the federation level, the fDSB had to be configured to handle SSH message tunneling and forwarding from the federation to cluster nodes, passing through the INRIA SSH gateway.

#### Amazon EC2

Rented Amazon EC2 instances also integrate SOA4All testbed. In order to simplify the inclusion of Amazon EC2 instances, a special Amazon Machine Image (AMI) was prepared including software and configuration required for the execution of Petals DSBs and the fDSB.

Amazon offers a range of instances with different amount of memory, CPU and I/O performance and pricing. The amount of CPU that is allocated to a particular instance is

expressed in terms of these EC2 compute units (according to Amazon, one EC2 compute unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. I/O only presents an indicator and can be *moderate* or *high* or *very high*.

Two of the most used Amazon EC2 instances were used in fDSB experiments:

- Small Instance has 1.7 GB memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB instance storage (150 GB plus 10 GB root partition), as a moderate I/O performance and is a 32 bit platform
- High-CPU Extra Large Instance has 7 GB of memory, 20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each), 1690 GB of instance storage, high I/O performance and is a 64 bit platform.

Amazon EC2 allows users to define custom network configuration, which may include firewall and NAT configuration. Connection to Amazon EC2 domain is, therefore, straightforward because there is no special restriction on the usage of resources. Since public IPs are available under payment of a fee, we rented a public IP address and associated it to one of the Amazon EC2 instances, which acts as an endpoint to the Amazon EC2 service park.

An fDSB router, deployed in Amazon EC2, was configured to access other service parks. No special configuration is required to access the Amazon EC2 service park. One of the parameters that influence performance experiments on this testbed is the AWS region to be used (e.g., Europe/Singapore/US). For benchmarking purposes, selecting a different region will produce different results.

### Grid'5000

The Grid'5000 is national French Grid platform. It gathers 9 sites geographically distributed in France featuring a total of 5000 processors. To form our testbed, we selected three clusters with different performances over two Grid5000 sites: two of them at INRIA Sophia Antipolis and the other at INRIA Lille

- INRIA Sophia-Antipolis Suno cluster: composed by 45 nodes, interconnected through a Gigabit Ethernet network. CPU of suno cluster is the quad-core Intel Xeon E5520 (Xeon Nehalem) and 32 GB of memory.
- INRIA Sophia-Antipolis Azur cluster: composed by 49 nodes, interconnected through a Gigabit Ethernet network. CPU of azur cluster is the AMD Opteron 246 (with 2 cores) and 2 GB of memory.
- INRIA Lille Chuque cluster: composed by 52 nodes, interconnected through a Gigabit Ethernet network. CPU of chuque cluster is the AMD Opteron 248 (with 2 cores) and 4 GB of memory.

The different Grid5000 sites are connected through the Renater-4 dark fiber backbone, connected to the same VLAN at 10Gbps speed.

Regarding fDSB configuration, Grid5000 is more complex than the other platforms, because machines are completely isolated from the Internet. Therefore, DSB nodes running in Grid5000 can only be accessed by the fDSB through SSH message tunneling and forwarding. The same is required for nodes to contact the fDSB.

## 2.2 Web Service Generation

### 2.2.1 Genesis

GENESIS<sup>1</sup> has been developed to solve a major problem in the current state of the art of software development for Service-oriented Architecture (SOA). So far, software testing in the SOA domain has been mostly concentrated on checking individual Web services regarding their performance, stability, fault tolerance, and other quality attributes. In our opinion not enough effort has been invested into supporting the testing of complex SOA components, which operate on (possibly large-scale) service-based environments

GENESIS [1] was introduced and described in detail in deliverable D1.5.1. Currently, a new version is being developed by the Vitalab group, but it's not available yet for download (promised release date by end of 2010). For the new version of GENESIS high priority has been assigned to a seamless extensibility of the framework in order to emulate arbitrarily structured testbeds composed of diverse SOA components, and to program their behavior.

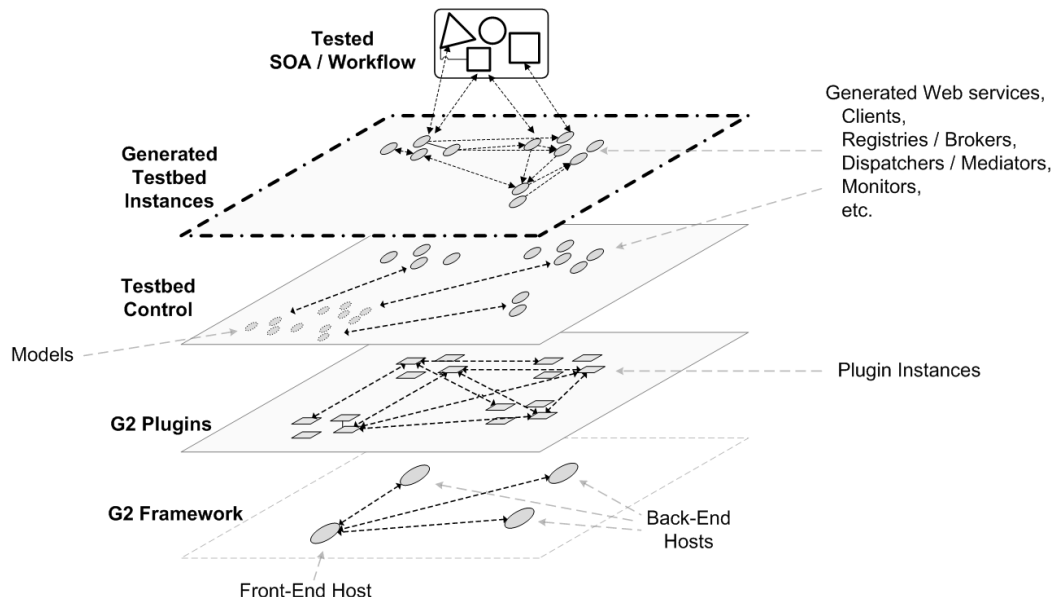


Figure 1: GENESIS Architecture

REST services support is still missing, so the need for an extension is still required. The following sections describe the work performed by TXT to design and to develop an **extension to GENESIS** in order to support **REST services generation**.

### 2.2.2 REST Services Support for Genesis

The main goal of the REST extension for GENESIS is to self generate/simulate new REST services, based on a similar approach to the existing WSDL Services generation in GENESIS

The activities performed in order to extend the platform are the following:

- Study of the existing Genesis architecture

<sup>1</sup> <http://www.infosys.tuwien.ac.at/prototype/Genesis>

- Definition of new required features
- Definition of a new technical architecture
- Development of the required extensions

More in detail the work performed on the GENESIS platform is the following:

1. Modification of the Genesis configuration file, in order to let end-users specify desired REST resources
2. Modification of the Genesis classes to parse and to process such new configuration file
3. Modification of the Genesis classes to self-generate WADL files out of the information provided in the configuration file
4. Modification of the Genesis classes to self-generate REST services based on the WADL file
5. Modification of the Genesis classes to self-deploy REST services based on the information provided in the configuration file

Thanks to such extensions, Genesis can now support both WSDL and REST services.

The following sequence diagram shows how Genesis can generate REST services (RS):

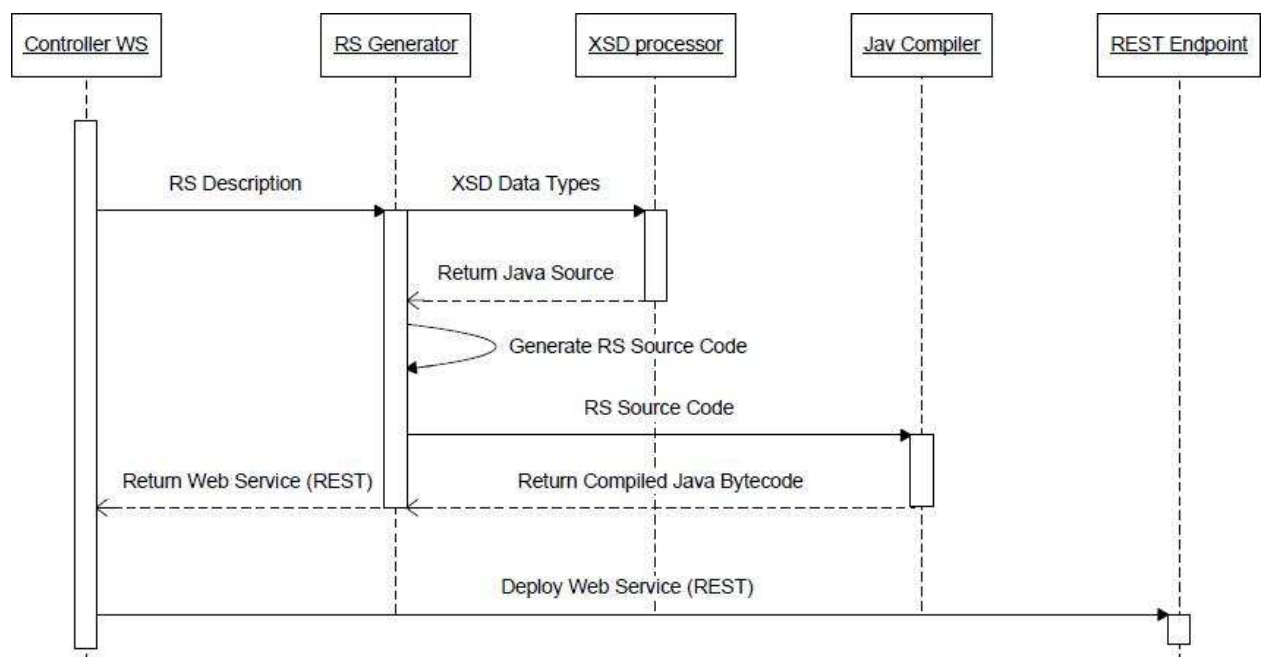


Figure 2: REST Service Generation with GENESIS

### 2.2.3 Technical Implementation

#### Genesis New Architecture:

The following picture shows the modified logical architecture of GENESIS:

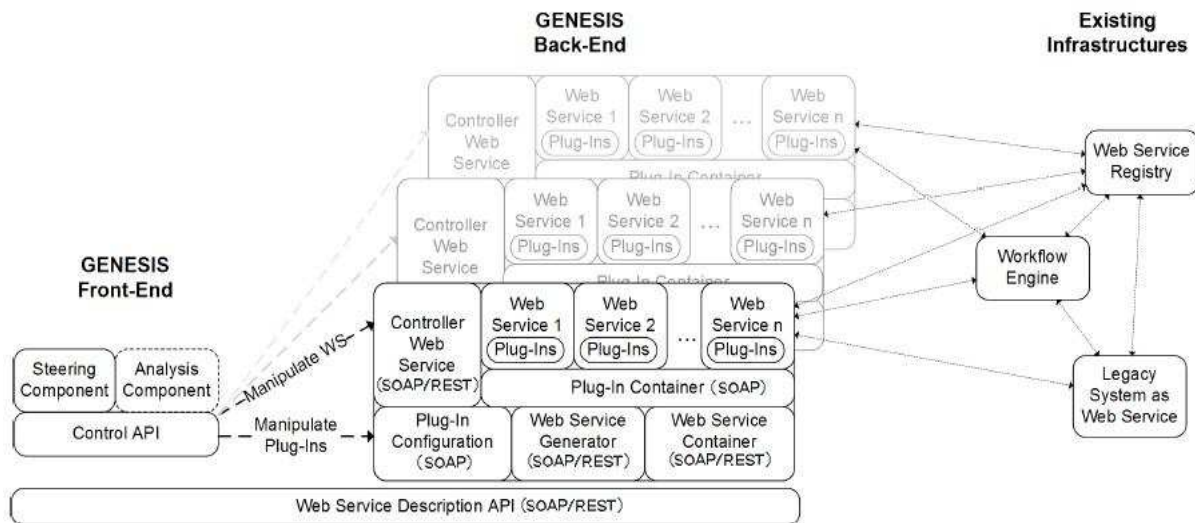


Figure 3: Genesis for REST logical architecture

### Genesis Configuration File:

The Genesis configuration file is the starting point to build REST services. For the definition of SOAP services, the existing configuration file uses the XML element “service” as child of “host” element; to configure a REST service the new XML element to use is: “<application>”.

Once that a host element has been created and its address has been defined, it is possible to build the REST service. In contrast to SOAP services, there is no one endpoint per service, so each host can contain a maximum of one REST service. Furthermore, each REST service (or application) can have unlimited resources, so inside the “application” element we can define several <resource> elements corresponding to all the resources we need using a different *path* for each one of them.

Complex types can be defined in two ways: in an external “.XSD” file to import or inline, inside the “<schema>” element.

To define a new method, it’s necessary to add the XML element “<method>” as child of the element “<resource>”; it’s mandatory to specify the HTTP name of the method in the attribute “name” which can be POST / GET / PUT / DELETE and it is necessary to define an “id” for the method.

To set the **input parameters** of a method just add the child element “<input>” inside the element “<method>” then add parameters inside “input”, using the attribute “type” to set the input type of your parameter.

In case of GET methods, it is necessary to set the output result by adding the child element “<output>” inside the element “<method>”. As for input parameter, use attribute “type” to set the return type of the method.

The attribute “**param-type**” specifies the type of the input parameter: possible values are: path, query, matrix, header, cookie, form. All of these values denote different possibilities to provide input data to the REST service.

Finally, the attribute “**path**” is used to define a sub-resource

By default, the configuration file that Genesis automatically reads is **configuration.xml**

located in directory: */conf*.

When we define a new host in the configuration file, by default only 8060, 8070 or 8080 ports are allowed.

### **Classes:**

New classes have been created inside Genesis, while other existing have been modified. The most relevant changes are in package: [at.ac.tuwien.vitalab.genesis.model](http://at.ac.tuwien.vitalab.genesis.model). They are represented in the following class diagram:

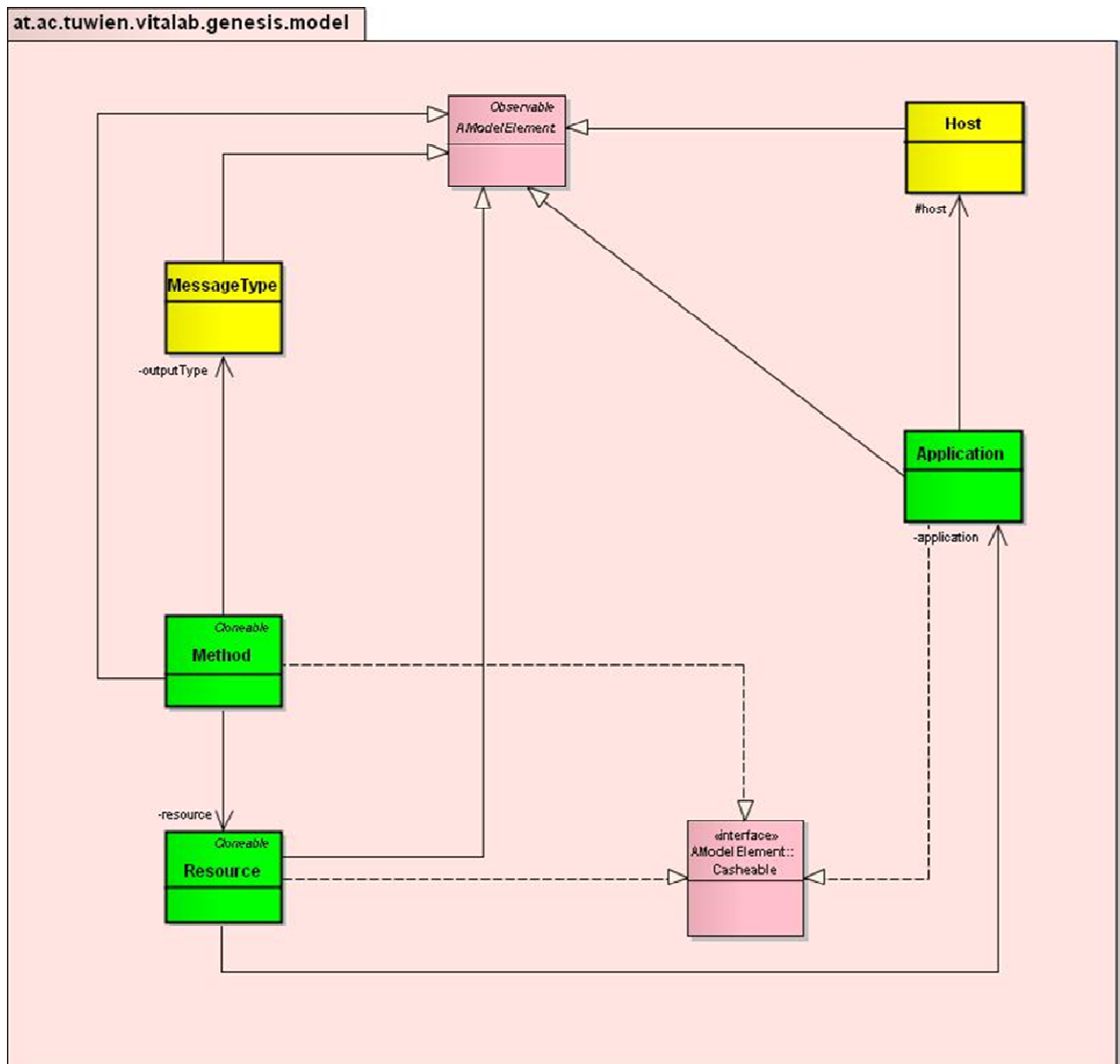


Figure 4: Diagram of new classes for package `at.ac.tuwien.vitalab.genesis.model`

**Package:** `at.ac.tuwien.vitalab.genesis.model`

**Added Classes:**



- **Application** : Is the model Class that contains the information about an *Application* (REST Service), starting from the definition in the configuration file.
- **Resource** : Is the model Class that contains the information about a *Resource*.
- **Method**: Is the model Class that contains the information about a resource's *Method*.

**Updated Classes:**

- **Host**: Is the model Class that contains the information about an *Host*. This class has been modified to include not only SOAP Web Services (Service) but also REST Web Services (Application)
- **MessageType**: Is the model Class that maps XML Schema types to Java types. This class has been modified to provide the XML response of a GET method.

**Package:** [at.ac.tuwien.vitalab.genesis.server](http://at.ac.tuwien.vitalab.genesis.server)

**Added Classes:**

- **AWebApplication**: it is responsible of deployment and undeployment of a REST Service (Web Application). Here is where the REST endpoint is created.
- **AWebApplicationGenerator**: generate and compile the java source code of the REST Service (Web Application)

**Updated Classes:**

- **GeneratorService**: this class has been modified to include the generation of a REST Service (Web Application)
- **AWebServiceGenerator**: this is the old "Generator" class, it has only been renamed to remark the contrast with AWebApplicationGenerator
- **GeneratorConfig**: added the logic to work with REST service

**Package:** [at.ac.tuwien.vitalab.genesis.server.jaxws](http://at.ac.tuwien.vitalab.genesis.server.jaxws)

**Added Classes:**

- **DeployApplication**: added to enable the deploying of REST services
- **DeployApplicationResponse**: added to provide the Response for DeployApplication invocation
- **UndeployApplication**: added to enable the undeploying of REST services
- **UndeployApplicationResponse**: added to provide the Response for UndeployApplication invocation
- **ListApplications**: added to enable the listing of REST services
- **ListApplicationsResponse**: added to provide the Response for ListApplications invocation

**Package:** [at.ac.tuwien.vitalab.genesis.client.jaxws](http://at.ac.tuwien.vitalab.genesis.client.jaxws)

**Updated Classes:**

- **Genesis**: added 3 Web Method to enable Genesis to work with REST services

## 2.2.4 Installation

The modified GENESIS code is located here:

<https://svn.sti2.at/soa4all/trunk/etc/GenesisREST.zip>

To install it, just unzip the file.

- Pre-requisite: Apache ANT, JRE 1.6 or higher.

## 2.2.5 An Example

The first step is to create the configuration file: we start by defining a new “application” called “DemoApplication” with two resources, as shown:

```
<configuration>
  <environment>
    <host address="http://localhost:8070/WebServices/GeneratorService">
      <application name="DemoApplication">
        <resource name="CustomerResource" path="customer">
        </resource>
        <resource name="ItemResource" path="item">
        </resource>
      </application>
    </host>
  </environment>
</configuration>
```

We then create two complex types, defining them inline:

```
<schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="person">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="surname" type="xs:string"/>
      <xs:element name="zip" type="xs:long" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="item">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="cost" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
</schema>
```

These complex types, can be used as input parameter or output result of our methods.



We create a POST method called “addCustomer”, with “person” as input parameter:

```
<method name="POST" id="addCustomer">
  <input>
    <data type="person"/>
  </input>
</method>
```

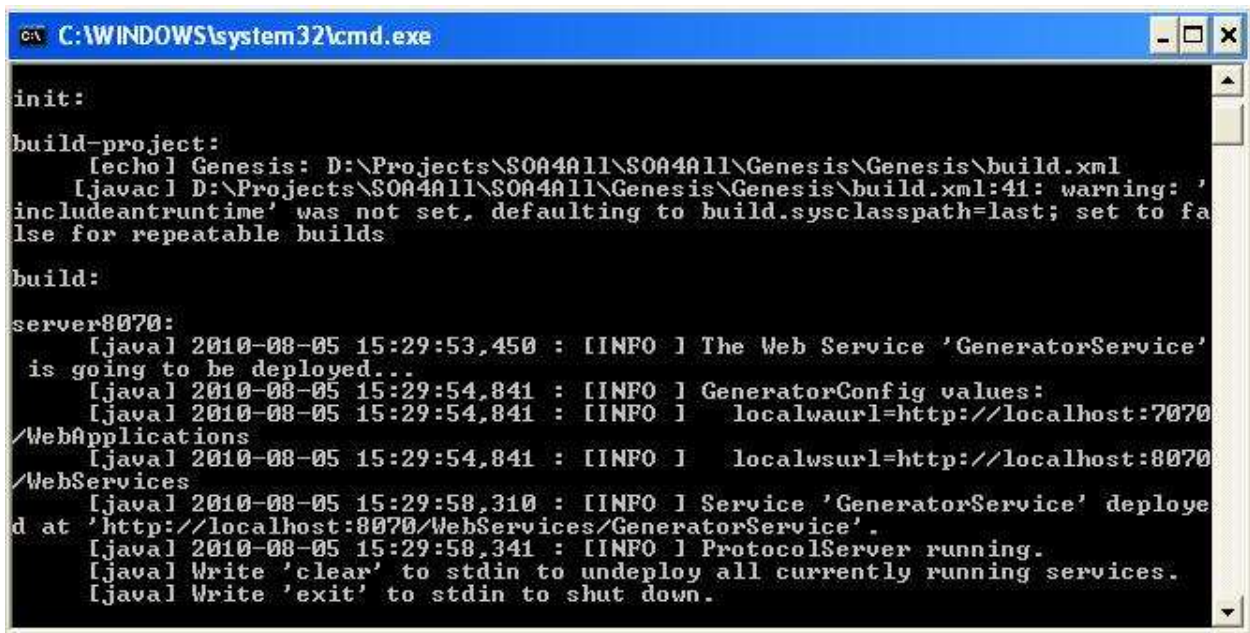
We also define a GET method called “getCustomer”

```
<method path="id" name="GET" id="getCustomer">
  <input>
    <id type="xs:string" param-type="path"/>
  </input>
  <output type="person"/>
</method>
```

Next step is to launch the appropriate “GeneratorService”. There are three “.bat” files to start three different Generators; you have to start only the ones described in your configuration file.

- If you have an host with the address:
  - <http://localhost:8060/WebServices/GeneratorService>  
you have to run: **ant8060.bat**
- If you have an host with the address:
  - <http://localhost:8070/WebServices/GeneratorService>  
you have to run: **ant8070.bat**
- If you have an host with the address:
  - <http://localhost:8080/WebServices/GeneratorService>  
you have to run: **ant8080.bat**

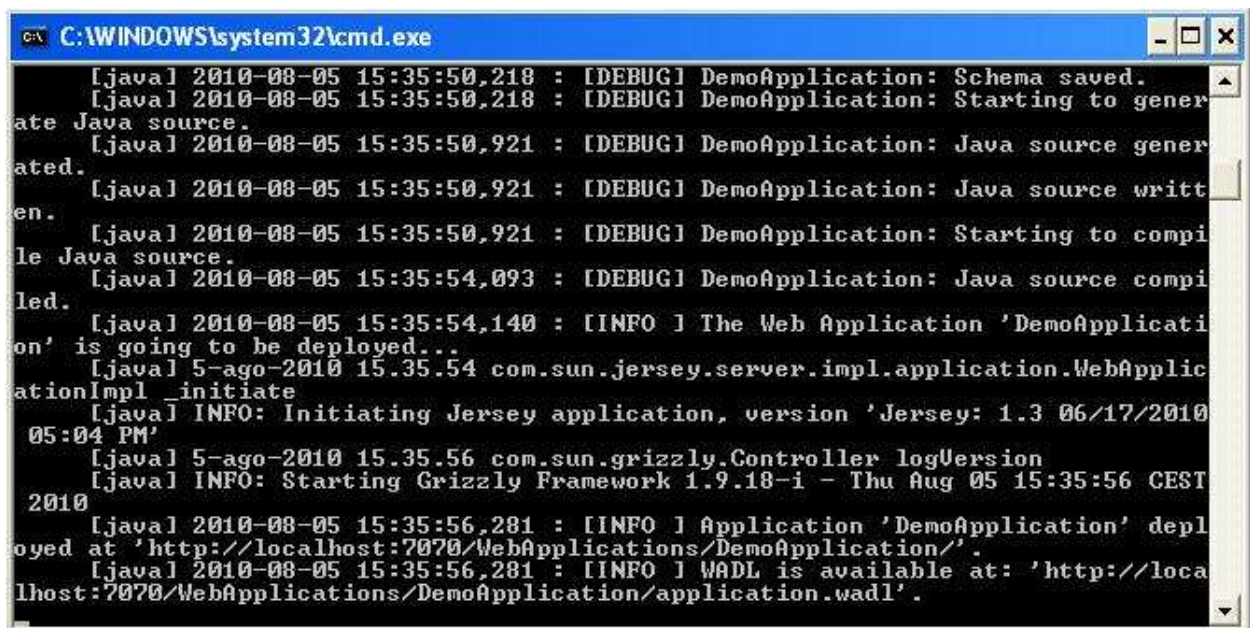
For each of the “.bat” files that you have launched, wait until the shell shows something like this:



```
C:\WINDOWS\system32\cmd.exe
init:
build-project:
[echo] Genesis: D:\Projects\SOA4All\SOA4All\Genesis\Genesis\build.xml
[javac] D:\Projects\SOA4All\SOA4All\Genesis\Genesis\build.xml:41: warning: '
includeantruntime' was not set, defaulting to build.sysclasspath=last; set to false for repeatable builds
build:
server8070:
[javal] 2010-08-05 15:29:53,450 : [INFO ] The Web Service 'GeneratorService'
is going to be deployed...
[javal] 2010-08-05 15:29:54,841 : [INFO ] GeneratorConfig values:
[javal] 2010-08-05 15:29:54,841 : [INFO ]   localwaul=http://localhost:7070
/WebApplications
[javal] 2010-08-05 15:29:54,841 : [INFO ]   localwsurl=http://localhost:8070
/WebServices
[javal] 2010-08-05 15:29:58,310 : [INFO ] Service 'GeneratorService' deployed
at 'http://localhost:8070/WebServices/GeneratorService'.
[javal] 2010-08-05 15:29:58,341 : [INFO ] ProtocolServer running.
[javal] Write 'clear' to stdin to undeploy all currently running services.
[javal] Write 'exit' to stdin to shut down.
```

Figure 5: Execution of .bat files

Now you can launch **deployment.bat**, to deploy your described services. If there's no error in your configuration file the generator service will start to generate then compile and finally deploy your services. If no errors occur, you will get something like this:



```
C:\WINDOWS\system32\cmd.exe
[javal] 2010-08-05 15:35:50,218 : [DEBUG] DemoApplication: Schema saved.
[javal] 2010-08-05 15:35:50,218 : [DEBUG] DemoApplication: Starting to generate
Java source.
[javal] 2010-08-05 15:35:50,921 : [DEBUG] DemoApplication: Java source generated.
[javal] 2010-08-05 15:35:50,921 : [DEBUG] DemoApplication: Java source written.
[javal] 2010-08-05 15:35:50,921 : [DEBUG] DemoApplication: Starting to compile
Java source.
[javal] 2010-08-05 15:35:54,093 : [DEBUG] DemoApplication: Java source compiled.
[javal] 2010-08-05 15:35:54,140 : [INFO ] The Web Application 'DemoApplication'
is going to be deployed...
[javal] 5-ago-2010 15.35.54 com.sun.jersey.server.impl.application.WebApplicationImpl
_initiate
[javal] INFO: Initiating Jersey application, version 'Jersey: 1.3 06/17/2010
05:04 PM'
[javal] 5-ago-2010 15.35.56 com.sun.grizzly.Controller logVersion
[javal] INFO: Starting Grizzly Framework 1.9.18-i - Thu Aug 05 15:35:56 CEST
2010
[javal] 2010-08-05 15:35:56,281 : [INFO ] Application 'DemoApplication' deployed
at 'http://localhost:7070/WebApplications/DemoApplication/'.
[javal] 2010-08-05 15:35:56,281 : [INFO ] WADL is available at: 'http://localhost:7070/WebApplications/DemoApplication/application.wadl'.
```

Figure 6: Final deployment

The last lines will show you the URL of the REST Service (Application) and the URL of the WADL related to your REST Service.

The REST service is now ready to be used (by any REST client) for testing.

### 3. SOA4All Runtime Evaluation

This section is divided into two parts. In the first part we describe the different evaluation scenarios that were defined for the SOA4All runtime. The second part then summarises the various results collected during the execution of these scenarios.

#### 3.1 Evaluation Scenarios

The scenarios utilize the SOA4All testbed infrastructure that was described in the previous section. This section contains both scenarios concentrating on performance experiments of the fDSB itself, in particular its scalability regarding the number of deployed DSB nodes and services, and finally experiments for the Semantic Spaces solution developed within WP1.

##### 3.1.1 fDSB Evaluation

In order to evaluate the performance/scalability of the fDSB, we have performed a series of experiments involving the testbed describes in Section 2.1. Initially, we performed deployment experiments to verify DSB and fDSB integration. Then, we performed experiments to verify performance (and quantify overhead) of fDSB invocations in relation to local DSB invocations. In the next subsection, we present each of these experiments.

###### 3.1.1.1 fDSB Deployment and Integration

For testing the fDSB deployment and integration, we carried out a large deployment of more than 700 DSB nodes over the testbed resources. The federation deployment consisted of three service parks integrated by the fDSB infrastructure, deployed over Amazon EC2 resources, Grid5000 and an INRIA private cluster.

The following table shows the number of nodes involved:

	Physical nodes (cores)	DSB nodes / node	Total DSB Nodes
INRIA eon cluster	20	4	80
Grid5000	146	4	584
Amazon EC2	2 (instances)	2	4
TOTAL	168	-	668

*Table 1: Experimental fDSB Deployment Resources*

Deployment and connection times remained constant, despite the number of nodes involved. Simple invocations were already performed and performance also remained constant. Stress tests on large multidomain platforms are still to be done, but depend on the completion of SOA4All integration tasks and use cases. As stated in the introduction of this deliverable, the results of these additional stress tests will be reported in an update of this document.

### 3.1.1.2 fDSB Communication

Figure 7 shows the organization of deployments on the testbed infrastructure. This multi-domain deployment presents well-defined gateway nodes in each of the Service Parks. Thanks to the fDSB infrastructure and multi-protocol communication, every DSB node is logically connected to other federation nodes. For performance reasons, internal communication is done using Java RMI, and external communication using RMISSH (RMI tunneled through SSH connections). However, the transport protocol could be HTTP, SOAP, etc. as well. More information about the federation multiprotocol communication is available in [10].

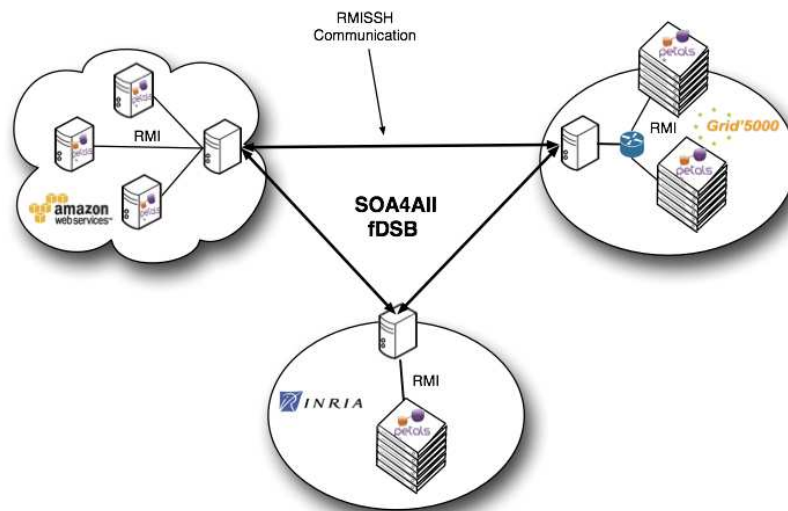


Figure 7: Testbed deployment

The service invocation experiments make use of a popular service benchmarking tool called SOAPUI [12]. SOAPUI was configured to perform service invocations over a local DSB and according to the test to be performed (local DSB communication or fDSB communication), services were either deployed locally or remotely.

Local service invocations and fDSB invocations follow different paths in the fDSB. While local services are invoked directly through the Petals transport layer, fDSB invocations travel through the fDSB transport layer. Figure 8 shows the path of invocations where the client is located in the same administrative domain as the INRIA DSB and the invoked Web Service is located on (i.e. bound to) an Amazon EC2 instance. Initially, a message is sent to the SOAP Binding Component of one of the INRIA DSB nodes and then it performs a lookup locally. If the service is not available in local DSB, a lookup process happens in the context of the federation by using the fDSB; the reply to this lookup is an endpoint which is not available locally; so, when sending a message to this endpoint, it will be forwarded to the federation, (the lookup query result is cached) and then sent through the fDSB transport to the Petals transporter of the other federated DSB; the last step consist in delivering the message to the real Web Service.

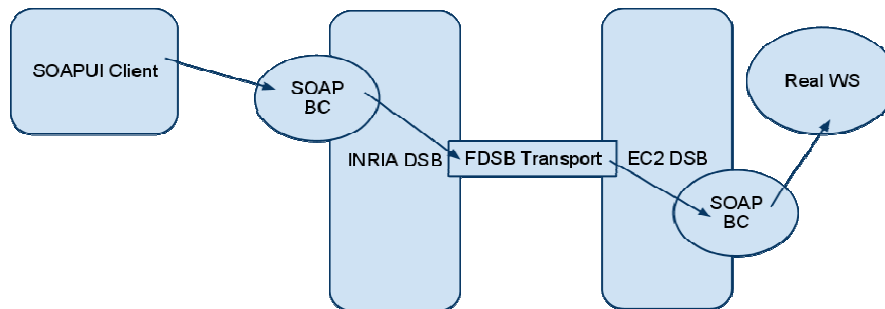


Figure 8: fDSB Service Invocation Path

In section 3.2.1, we present performance results of such invocations and analyze the result to verify the expected overhead of the fDSB layer.

### 3.1.2 Cloud Bursting Scenario

This scenario is a pure based scenario that involves different cloud infrastructures (at least 2). One is a private cloud within the company and an external cloud provider where our services can run and remain connected to the private cloud in a transparent way to the user.

In this scenario the DSB is deployed inside the private cloud of the company and, when the escalation of the VMs is taking place, this will not only be within the infrastructure of the cloud provider but also the service tested will spill over in a different cloud provider. This is called Cloud Bursting [13].

Originally the bursting technique was applied to keep a good bandwidth of a service, but it also can be applied to other parameters once we use it on the cloud. By playing with the SLA and the Cloud Bursting the private cloud would be an unlimited resource cloud in ideal conditions as you would rent resources from external cloud providers, saving money in infrastructure to the company. But there are some rules defined to perform the escalation and being able to define which services deploy on the external cloud provider and which don't regardless the external cloud policies or SLAs. Once we deploy our services in a different cloud, the interconnection can be defined also in our policies. For this the fDSB comes into the scene as we won't let any message going from the internal cloud to the external where the new service is deployed without going via the fDSB, whereas as said before within the private cloud infrastructure a normal DSB is used to perform the communication between the services.

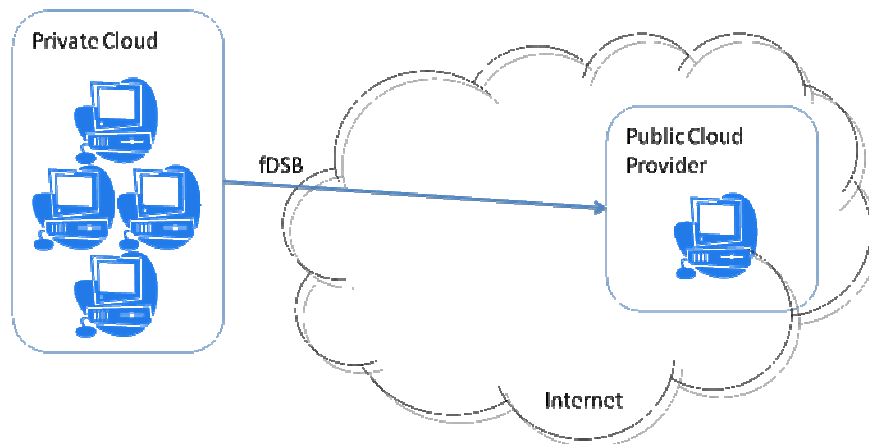


Figure 9 Cloud Bursting

### 3.1.2.1 Example scenario

When a company has its own cloud, this normally can handle with a certain amount of VMs running simultaneously with the efficiency that the company is expecting. Then, the cloud bursting is a good technique to get more resources.

Saying that for a specific service we have an SLA like this.

1. Performance < 0.5 sec.
2. Internal VMs <= 10.

These two rules would make our service scale horizontally. If we set the performance, like in rule 1 and we run stress test as the ones run in SOA4All this rule will trigger a new instance that would let us continue respect this SLA.

Also, in a period where our customers are using our service or our test have increased. Based on our own business rules or cost rules the company would not like to host more than a certain VMs for a single service and reached the limit imposed in the rule 2, the next escalation would take place in an external cloud provider making the use of the cloud bursting.

Therefore, as the fact of this intercommunication between the internal infrastructure within the company and an external infrastructure. The communication between the external VMs and the internal ones offering the service has to be done using the fDSB in order to preserve the privacy of the data sent by our customers.



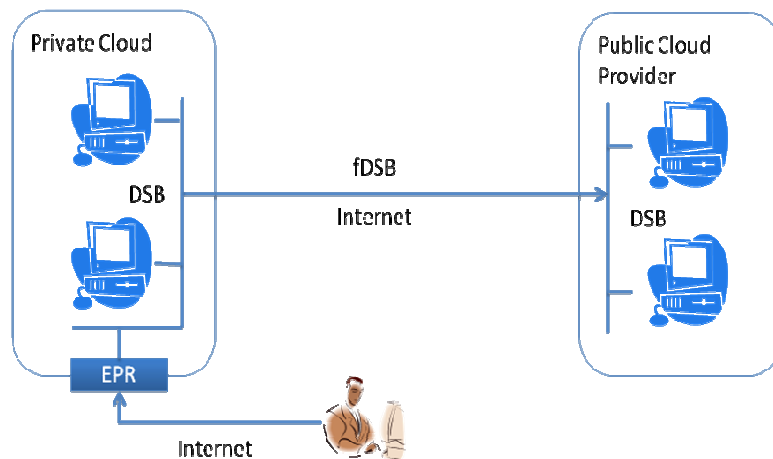


Figure 10: Cloud Bursting deployment

### 3.1.3 Distributed Space

In order to validate the distributed space architecture, we have performed extensive experiments. The goal was twofold. First, we wanted to evaluate the overhead induced by the distribution and the various software layers lying between the repository and a user. Second, we wanted to evaluate the benefits of our approach, namely the scalability in terms of concurrent access and overlay size.

- Insertion of random data: the first set of experiments inserts 1000 randomly generated statements in an overlay made of 1 to 100 peers.
- Queries using BSBM data: to evaluate distributed queries, we have used a subset of the BSBM benchmark to generate meaningful data and queries. These experiments have been performed with 100 peers.

## 3.2 Runtime Evaluation Results

This section summarises the results from the evaluation process of the different parts of the SOA4All runtime – more specifically the fDSB, as well as the Semantic Spaces. A variety of experiments have been conducted on the different testbeds, the results of which are reported below. A comparison to an alternative solution, similar in scope to the functionalities offered by the fDSB is included as well, in order to provide a context to the evaluation data. Finally, the results for the experiments of the Semantic Spaces solution developed in WP1 have been published as a paper, with parts of the findings reported in this section. The complete paper has been attached to this deliverable.

### 3.2.1 fDSB Evaluation

As explained in section 3.1.1.2, we developed and executed various experiments in the SOA4All testbed, presented in section 2.1. The main goal of this evaluation is to compare performance of service invocations between different service parks and local service invocations.

Table 2 summarizes the average invocation times in the different configurations. The first column indicates the origin of the service invocations and the other columns the destination. Cells intersecting the same domains (e.g. INRIA-INRIA) present the times for local invocations without and with the fDSB, respectively.

Comparing local invocations with and without the federation, we notice the overhead is about 14% in average. Invocations between distant DSBs are naturally slower than invocations between local DSBs because they go through the Internet, passing through gateway nodes. In the best case (between INRIA and Grid 5000, due to the fact that they are in the same Internet backbone) the average overhead was 7.3%, while in the worst case the overhead was 107.5% in relation to local fDSB calls. These invocation have been between Grid5000 and Amazon EC2, which goes through Internet between France and US. While these numbers are not negligible, they are still unavoidable, considering that the service invocations happen across the Internet.

Origin \ Dest.	INRIA	Grid5000	Amazon EC2
INRIA	45.2 / 51.5	55.3	106.95
Grid5000	57.4	27.9 / 31.5	108.4
Amazon EC2	113.03	104.4	54.21 / 62.3

Table 2: fDSB Average Invocation Times

### 3.2.2 Semantic Spaces evaluation

#### 3.2.2.1 Insertion of random data, single peer

The first experiment performs 1000 statements insertion and measure the individual time for each of them, on a CAN made of a single peer. The two entities of this experiment, the caller and the peer, are located on the same host. The commit interval was set to 500ms (TODO: explain) and 1000 random statements were added. Figure 11 shows the duration of each



individual call. On average, adding a statement took 1.853ms with slightly higher values for the first insertions, due to cold start.

In a second experiment, the caller and the peer were put on separate hosts to measure the impact of a local network link on the performance. As shown in Figure 12, almost all add operations took less than 5ms while less than 2% took more than 10ms. The average duration for an add operation was 5.035ms.

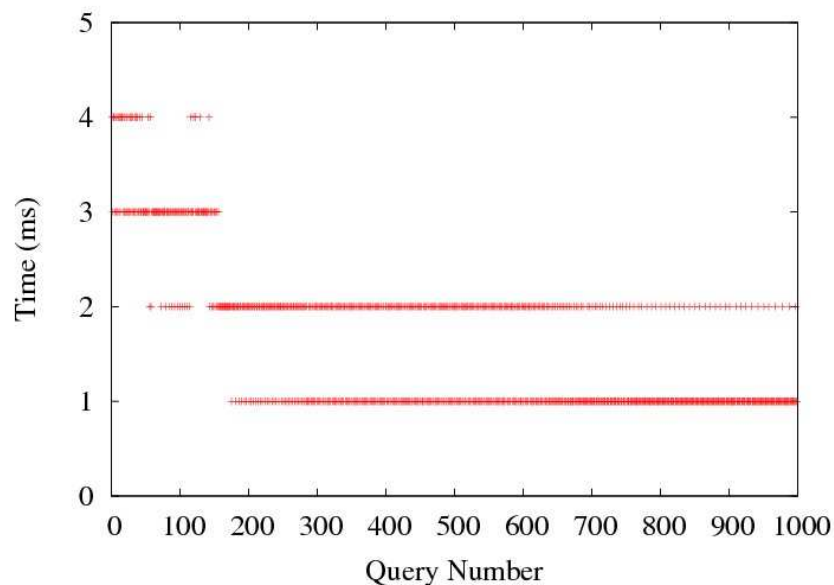


Figure 11: Individual time for sequential insertion of random statements on a single local peer

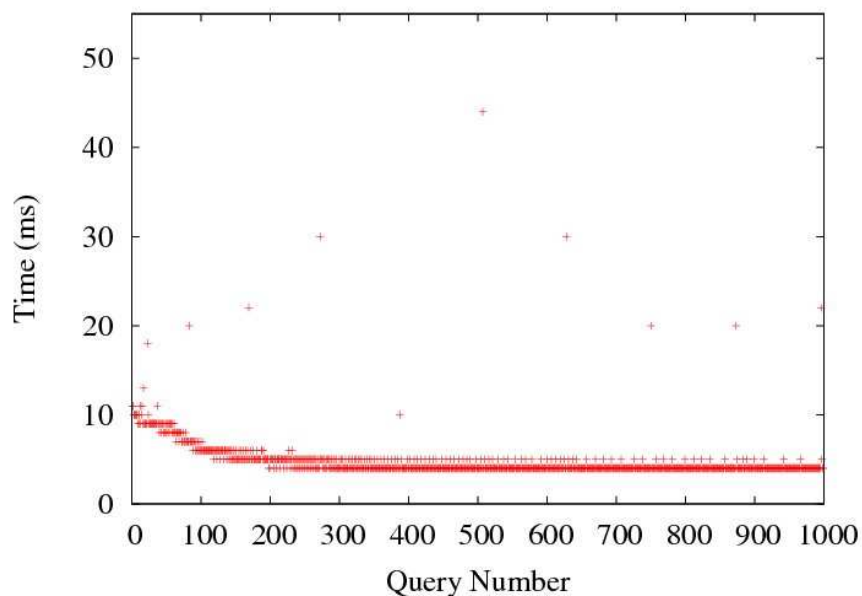


Figure 12: Individual time for sequential insertion of random statements on a remote peer

### 3.2.2.2 Insertion of random data, multiple peers

We have measured the time taken to insert 1000 random statements in an overlay with different number of peers, ranging from 1 to 100. Figure 13 shows the overall time when the calls are performed using a single (left) or 32 threads (right). As expected, the more peers, the longer it takes to add statements since more peers are likely to be visited before finding

the correct one. However, when performing the insertion concurrently, the total time is less dependent on the number of peers. Depending on the zones various sizes and the first peer randomly chosen for the insertion, the performance can vary, as can be seen with the 50 peers experiments.

To measure the benefits of concurrent access, we have measured the time to add 1000 statements on a 100 peers overlay, varying the number of threads from 1 to 30. Results in Figure 14 show a sharp drop of the total time, clearly highlighting the benefits of concurrent access.

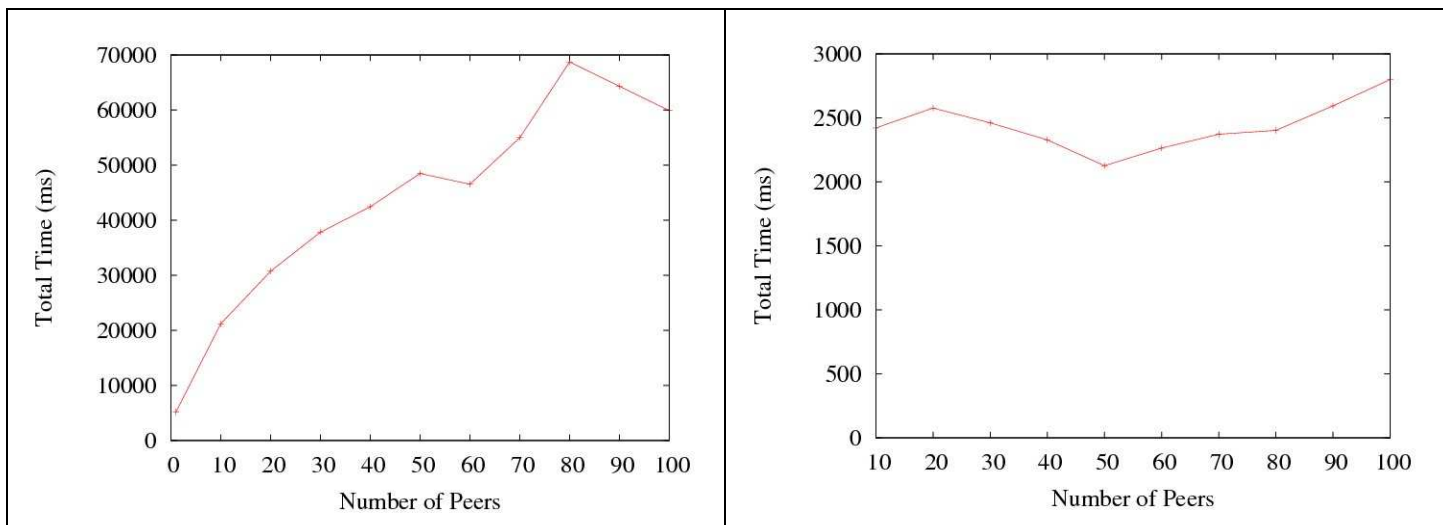


Figure 13: Insertion of 1000 statements for variable number of peers, 1 thread (left) and 32 threads (right)

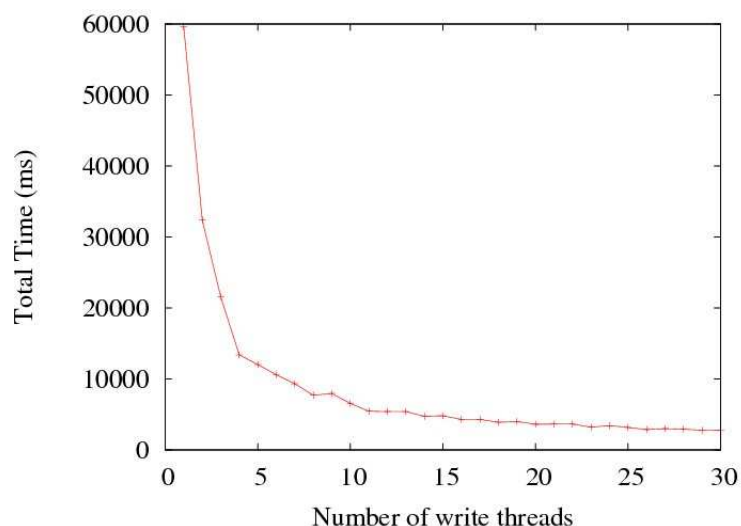


Figure 14: Evolution of the time for concurrent insertion on a 100 peers overlay

### 3.2.2.3 Queries using BSBM data

The *Berlin SPARQL Benchmark* (BSBM) [9] defines a suite of benchmarks for comparing the performance of storage systems across architectures. The benchmark is built around an e-commerce use case in which a set of products is offered by different vendors, and consumers have posted reviews about products. The following experiment uses BSBM data with custom queries detailed below. The dataset is generated using the BSBM data generator for 10 products. It provides 4971 triples which are organized following several categories:

- 289 Product Features
- 1 Producer and 10 Products
- 1 Vendor and 200 Offers
- 1 Rating Site with 5 Persons and 100 Reviews.

The queries use the following prefixes:

```
PREFIX bsbm: http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/
PREFIX bsbm-ins: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iso: http://download.org/rdf/iso-3166/countries#
PREFIX purl: <http://purl.org/stuff/rev#>
```

During these experiments, we have used the following queries:

Q1 : Returns a graph where producers are from Deutschland

```
CONSTRUCT {
  iso:DE <http://www.ecommerce.com/Producers> ?producer
} WHERE {
  ?producer rdf:type bsbm:Producer.
  ?producer bsbm:country iso:DE
}
```

Q2: Returns a graph with triples containing instances of Review

```
CONSTRUCT {
  ?review rdf:type purl:Review
```

```

} WHERE {
  ?review rdf:type purl:Review
}

```

Q3 Returns a graph where triples imply a *rdf:type* relation as predicate

```

CONSTRUCT {
  ?s rdf:type ?o
} WHERE {
  ?s rdf:type ?o
}

```

Q4] Returns a graph where *bsbm-ins:ProductType1* instance appears

```

CONSTRUCT {
  bsbm-ins:ProductType1 ?a ?b.
  ?c ?d bsbm-ins:ProductType1
} WHERE {
  bsbm-ins:ProductType1 ?a ?b.
  ?c ?d bsbm-ins:ProductType1
}

```

Queries Q1 and Q4 are complex and will be decomposed into two sub-queries. Hence, we expect a longer processing time for them. The number of matching triples is the following:

Q1	Q2	Q3	Q4
1	100	623	7

Figure 15 shows the execution time and the number of visited peers when processing Q1, Q2, Q3 and Q4. Note that when a query reaches an already visited peer, we count it although it will not be further forwarded. Q1 is divided into two sub-queries with only a variable subject. Hence, it can efficiently be routed and is forwarded to a small number of peers. Q2 also has one variable and thus exhibits similar performance. Q3 has two variables so it will be routed along two dimensions on the CAN overlay, reaching a high number of peers. Since it returns 623 statements, the messages will carry a bigger payload than for the other queries. Finally, Q4 generates two sub-queries with two variables each, making it the

request with the highest number of visited peers. On the 100 peer network, the two sub-queries have visited more than 170 peers.

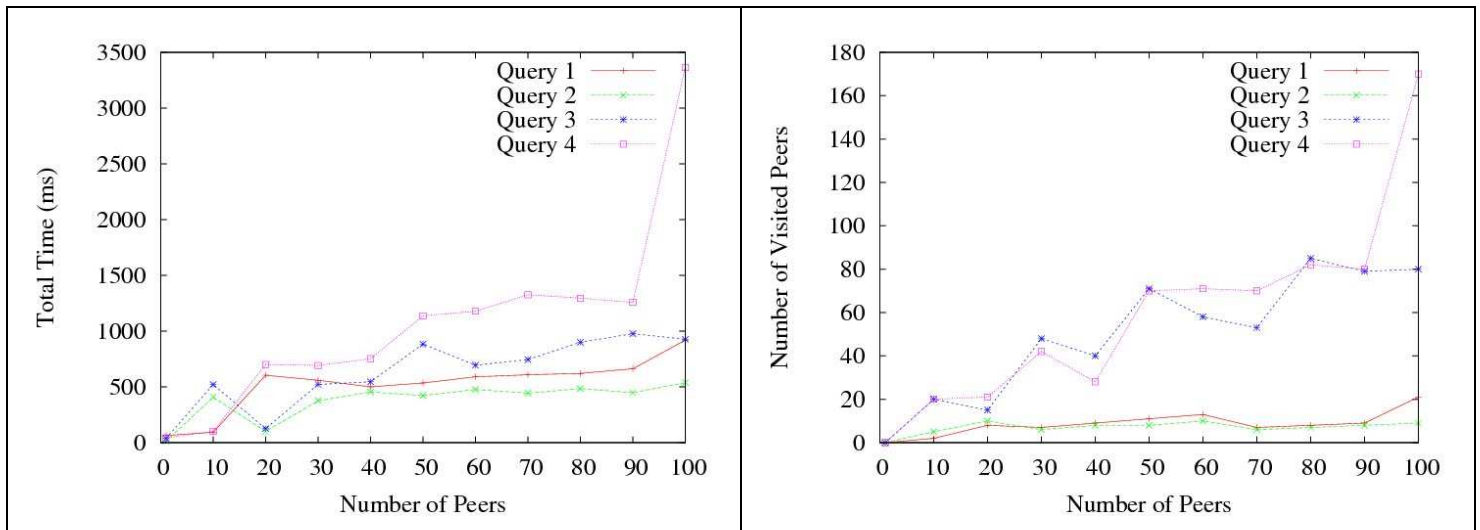


Figure 15: Custom queries with BSBM dataset on various overlays, execution time (left) and message overhead (right).

### 3.3. Comparison with other solutions

In this section we will describe a comparison of the SOA4All solution, based on the fDSB and Semantic Space, with a new one based on different products than the ones used for the development and implementation of the SOA4All runtime. The architecture suggested with this new solution will remain the same as we can see in Figure 7 with different organizations interconnected via a federated channel between them and with a DSB deployed within the organization infrastructure.

This solution can be applied to a cloud based service parks infrastructure based on VMs or over physical service parks as before and there is no need for them to be directly connected to internet as for the federation between the different infrastructures there will be a gateway to generate the trust circle through internet.

Let's focus on the infrastructure inside the companies. A set of service parks will be interconnected via a Service Bus that will be able to talk to all of them and the services deployed on them. There is the possibility of using the WSO2 ESB<sup>2</sup> that is an open source ESB. This ESB lets you to create internal endpoint references (EPRs) within the bus that can be used to balance the calls among different service parks. By running the different tests done above with the DSB used in SOA4All there won't be many differences in terms of speed as within one domain the speed is practically the same. However, this value can easily be affected by the different rules and policies that can be applied in the ESB, as this can be used to enforce the security or route a message based on its content extracting parameters or changing them once the message is inside the ESB.

<sup>2</sup> Fast, open-source ESB, based on Apache Synapse, available at <http://wso2.com/products/enterprise-service-bus/>

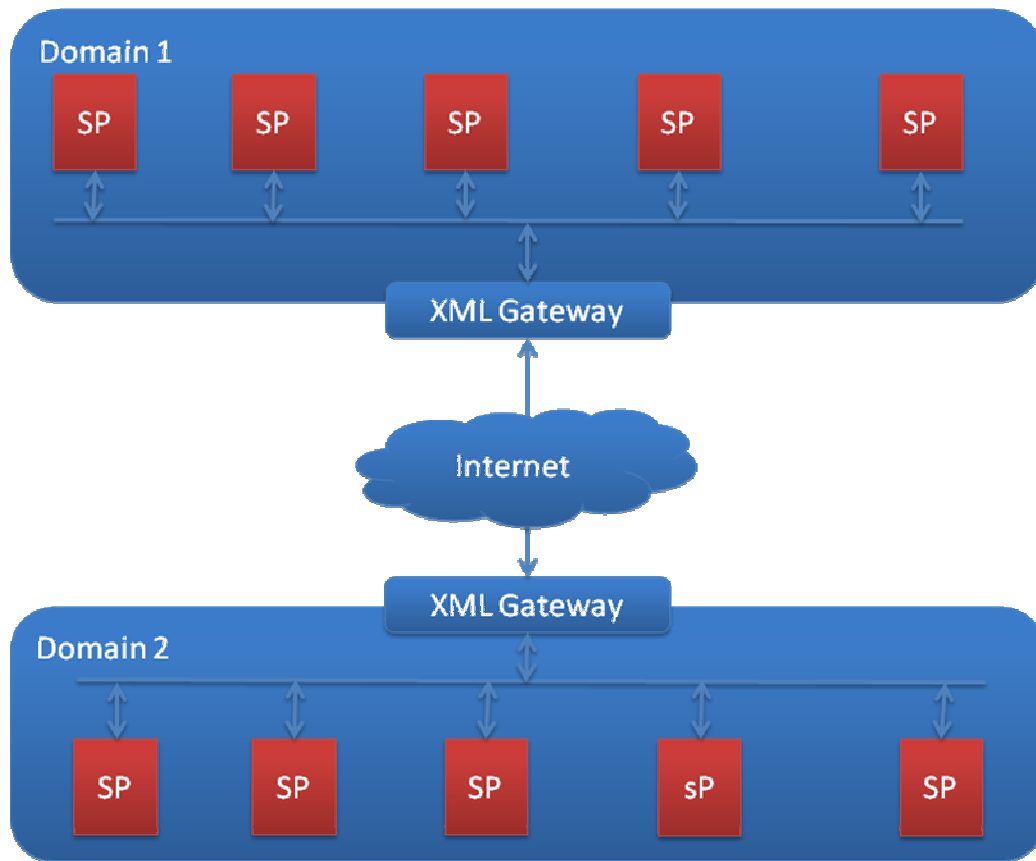


Figure 16 Gateways Scenario

In order to generate and maintain the federation between domains, a XML gateway will be in both sides of the domains, i.e. it will be the outbound gateway for the messages sent to other domains and it will be the inbound gateway where the messages will get through in order to pass to the internal EBS. In the XML Gateways there are private vendors that provide this solution such as (Vordel<sup>3</sup>, Layer7<sup>4</sup>, Cisco<sup>5</sup>, Forum<sup>6</sup>).

The given solution here is formed for at least 2 subsystems. The XML Gateway that will intercept the message, analyze it and perform some changes over it such as changing the destination or encrypting the message in order to enforce the security, and a SAML mechanism in order to sign the message before sending it to the other domain. By the combination of these 2 elements we can ensure the transport of the data between 2 domains and get a similar behavior as with the fDSB.

However even being this a possible solution to implement in the scenario, this will penalize the time of the operations, as the message has to go through many steps and call other services before it is sent to the destination domain. And it will make the communication slower than with the fDSB solution implemented in SOA4All.

<sup>3</sup> <http://www.vordel.com>

<sup>4</sup> <http://www.layer7tech.com>

<sup>5</sup> <http://www.cisco.com/en/US/products/ps6906/>

<sup>6</sup> <http://www.forumsys.com/products/xmlgateway.php>

## 4. Conclusions

In this deliverable, we have described the final setup of the testbed infrastructure environment for SOA4All. This infrastructure was used as part of the overall efforts to evaluate SOA4All project results. While the testbed infrastructure can be used by component owners, use case partners and dedicated testers to generate testbeds, create test cases and execute those test cases on the testbed, the main results of the evaluation efforts described in these deliverable focus on the performance and scalability testing of the technical artefacts developed in WP1 – i.e. the runtime environment.

We have described several evaluation scenarios which were implemented on the SOA4All testbeds, and have reported the results of conducting the experiments based on these scenarios. Finally, we investigated the possibility of using different technology than the one developed in SOA4All, which achieves similar functionalities, albeit at the cost of performance overheads.

While the experiments of the Semantic Spaces were concluded by M30 of the projects, the performance measurements for the fDSB are still ongoing, since we have created additional tests and experiments to be used on the complete integrated SOA4All platform. This includes services and tools from all technical work packages. A follow-up deliverable will therefore be made available by the end of the project, which collects the final results of the different evaluation efforts in the project and updates the relevant sections of this deliverable accordingly.

## 5. References

1. L. Juszczak, H.-L. Truong, and S. Dustdar, "Genesis - a framework for automatic generation and steering of testbeds of complex web services," in Proc. 13th IEEE International Conference on Engineering of Complex Computer Systems ICECCS 2008, March 31 2008–April 3 2008, pp. 131–140.
2. Schreder, B., Villa, M., Abels, S., Zaremba, M., Sheikhhasan, H., Puram, S.; Deliverable D9.2.1: eCommerce Framework Infrastructure Design, SOA4All: Service Oriented Architectures for All - 215219.
3. Vogel, J., Schnabel, F., Mehandjiev, N.; Deliverable D7.2 Scenario Definition, SOA4All: Service Oriented Architectures for All - 215219.
4. Lecue, F., Mehandjiev, N., Wajid, U., Namoune, A., Macaulay, L.; Deliverable D2.5.1: SOA4All Evaluation, SOA4All: Service Oriented Architectures for All - 215219.
5. Schreder, B., Cruz, S., Abels, S., Pariente, T., Richardson, M.: D1.5.1 SOA4All Testbeds Specification and Methodology, SOA4All: Service Oriented Architectures for All - 215219.
6. Schreder, B., Krummenacher, R., Abels, S., Pariente, T., Richardson, M., Villa, M., Di Matteo, G.: D1.5.2 Setup SOA4All Testbeds, SOA4All: Service Oriented Architectures for All - 215219.
7. Richardson, M., Davies, J., Stincic, S., Mehandjiev, N., Wajid, U., Lecue, F., Álvaro Rey, G.; Deliverable D8.3 Web21c Futures Design, SOA4All: Service Oriented Architectures for All - 215219.
8. Stinčić, S., Davies, J., Richardson, Álvaro Rey, G., Lecue, F., M., Mehandjiev, N., Maleshkova, M.; Deliverable D8.4 Web 21c Prototype v1, SOA4All: Service Oriented Architectures for All - 215219.
9. Christian Bizer and Andreas Schultz, The Berlin SPARQL Benchmark, 2009.
10. Hamerling, C., Legrand, V., Baude, F., et al. D1.4.1B SOA4All Runtime, 2009, SOA4All: Service Oriented Architectures for All - 215219.
11. Hamerling, C. Baude, F., Mathias E., et al. D1.4.2B SOA4All Runtime v2, 2010 (to appear), SOA4All: Service Oriented Architectures for All - 215219.
12. SOAPUI Web Service Testing. <http://www.soapui.org>, 2010.
13. Cloud Burst <http://en.wikipedia.org/wiki/Cloudburst>



## **Annex A.**

The paper “CAN-Based Approach for RDF Data Management in Structured P2P Systems” by I. Filali, L. Pellegrino, F. Bongiovanni and F. Huet has been attached to this deliverable.