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

D1.5.3B Testbeds Validation Update

Activity N:	1	
Work Package:	1	
Due Date:		29/04/2011
Submission Date:		28/04/2011
Start Date of Project:		01/03/2008
Duration of Project:		38 Months
Organisation Responsible of Deliverable:		HANIVAL
Revision:		2.0
Author(s):	Bernhard Schreder (Hanival), Juan Luis Prieto Martínez (ATOS), Matteo Villa (TXT), Giovanni Di Matteo (TXT), Claudio Stella (TXT), Fabrice Huet (INRIA), Elton Mathias (INRIA)	
Reviewers:	Alex Simov (Ontotext), Maurilio Zuccalà (CEFRIEL)	

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	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	
1.1	20/02/11	Add Test Plan Section	All
1.2	03/03/11	Updates to Section 3	Elton Mathias (INRIA)
1.3	20/04/11	Final updated version	Bernhard Schreder (Hanival)
2.0	26/04/2011	Final check for submission	Julia Wells (ATOS)

Table of Contents

List of Figures

List of Tables

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 test plans and validation processes used for performance testing of the SOA4All runtime environment. The test plan for each runtime component is described, along with any metrics suitable for the tests. The results of the tests for the fDSB and the semantic spaces have been collected in this deliverable as well 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 test plans and validation processes for the SOA4All runtime environment, as well as 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 test plans and evaluation scenarios used for performance testing of the SOA4All runtime environment. Each test plan focuses on a part of the overall runtime architecture, including the fDSB, semantic spaces, service location and service construction. The results of these tests have been collected in Section 4 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 evaluation of the SOA4All testbed infrastructure.

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 asses 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 entrypoint 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[1] 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.

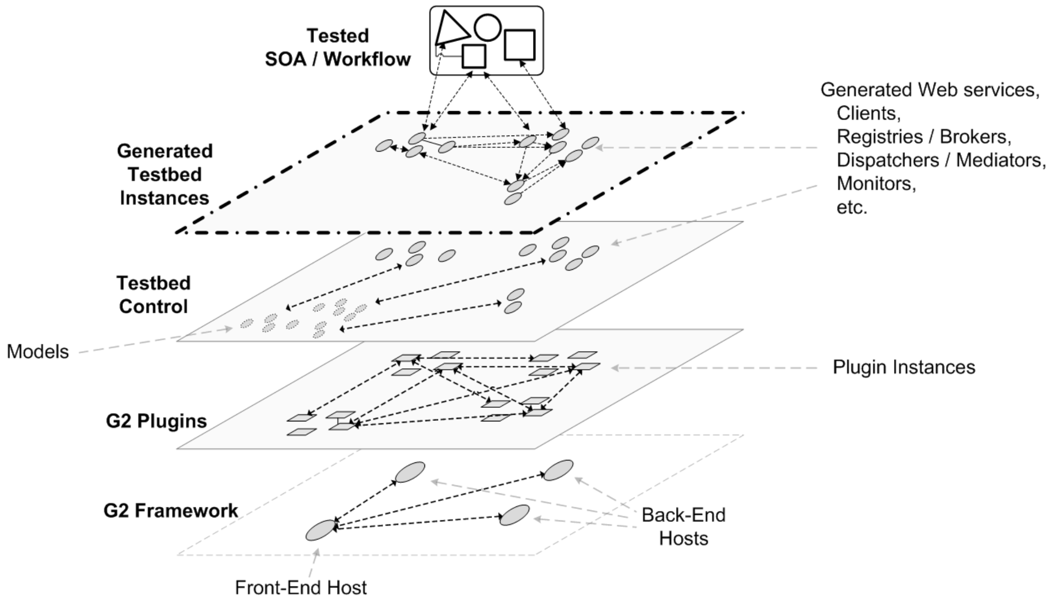


Figure 1: GENESIS Architecture

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

[1] 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):

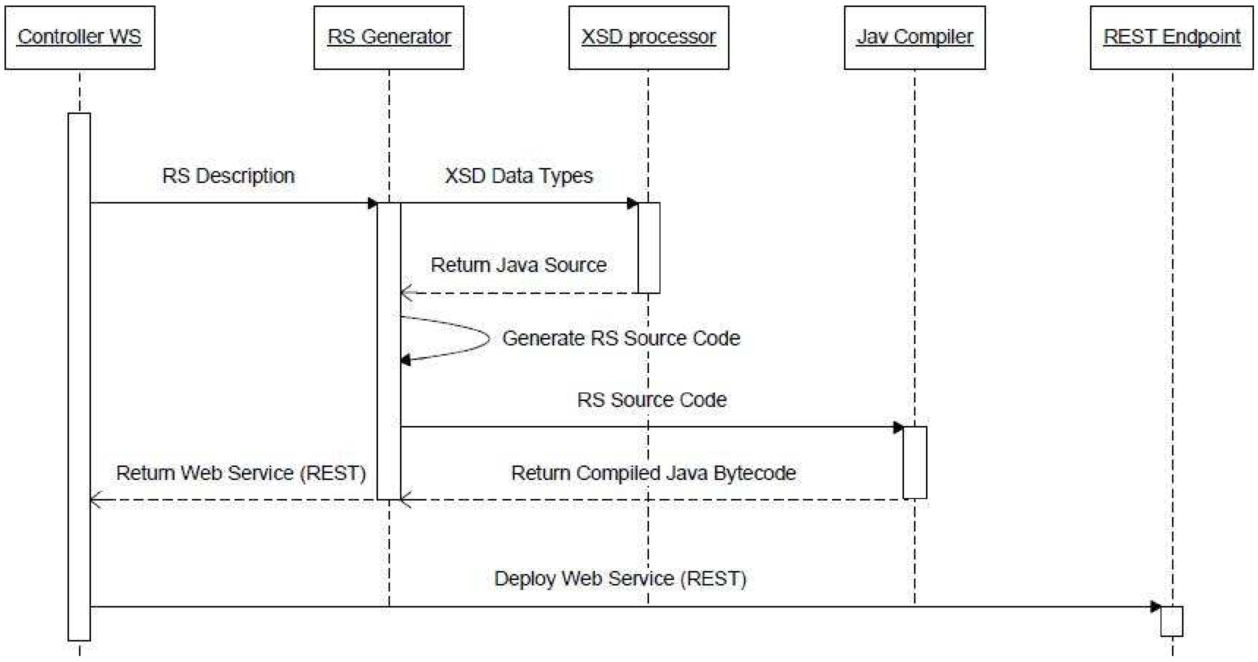


Figure 2: REST Service Generation with GENESIS

2.2.3 Technical Implementation

Genesis New Architecture:

The following picture shows the modified logical architecture of GENESIS:

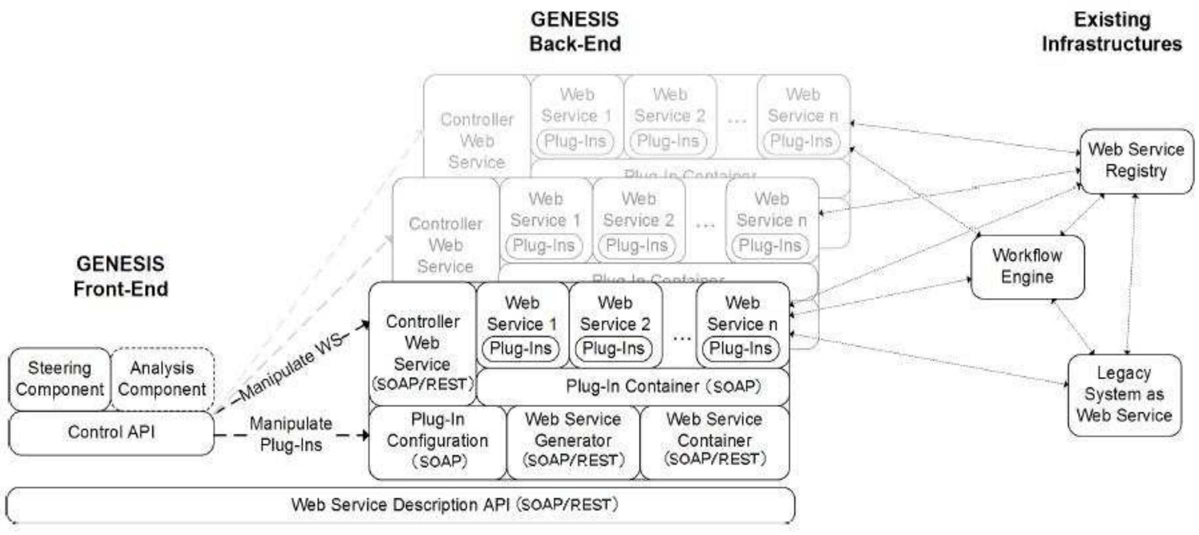


Figure 3: Genesis for REST logical architecture

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 differnet 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. They are represented in the following class diagram:

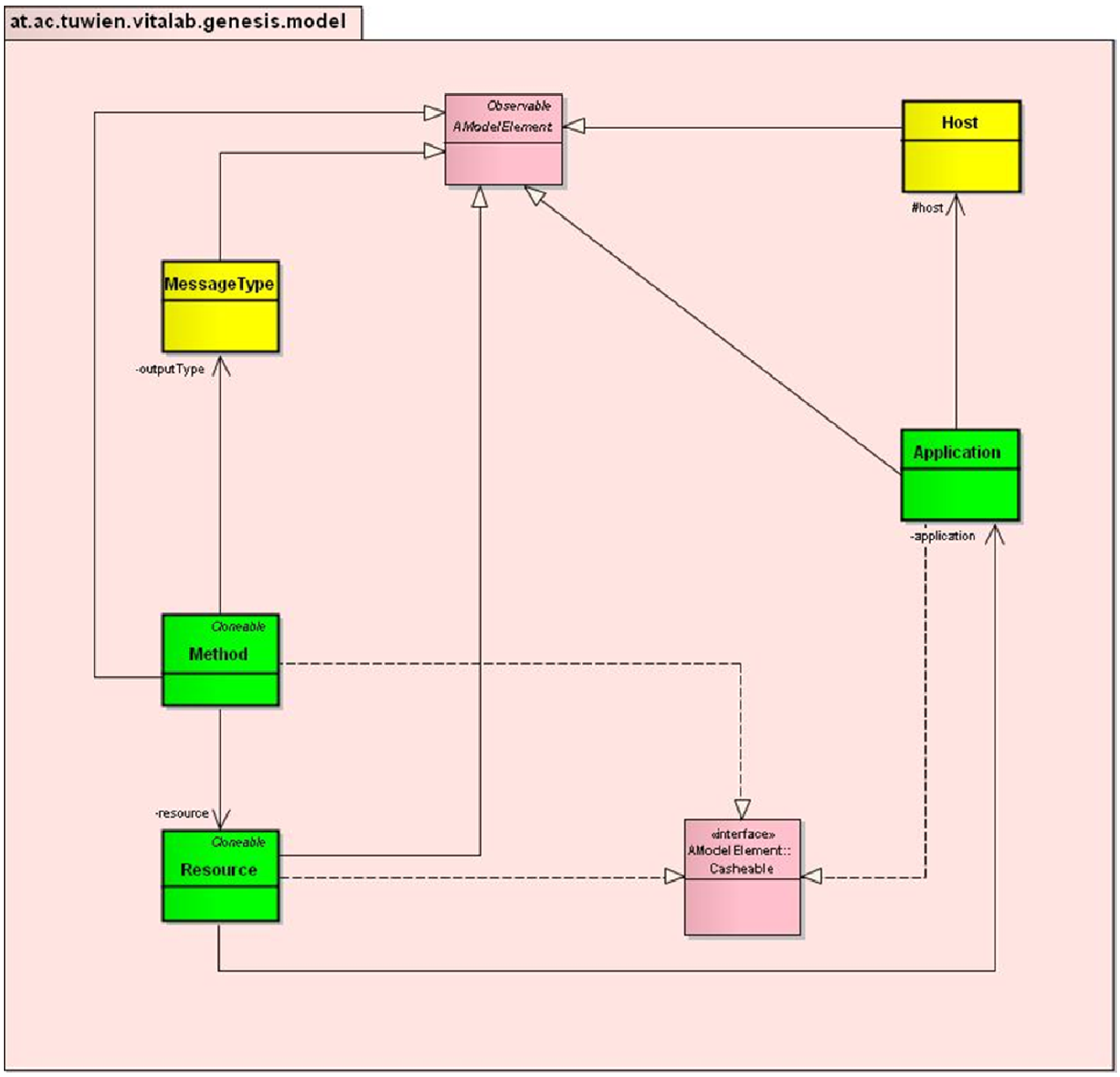


Figure 4: Diagram of new classes

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

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

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

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; only the ones described in the configuration file need to be started.

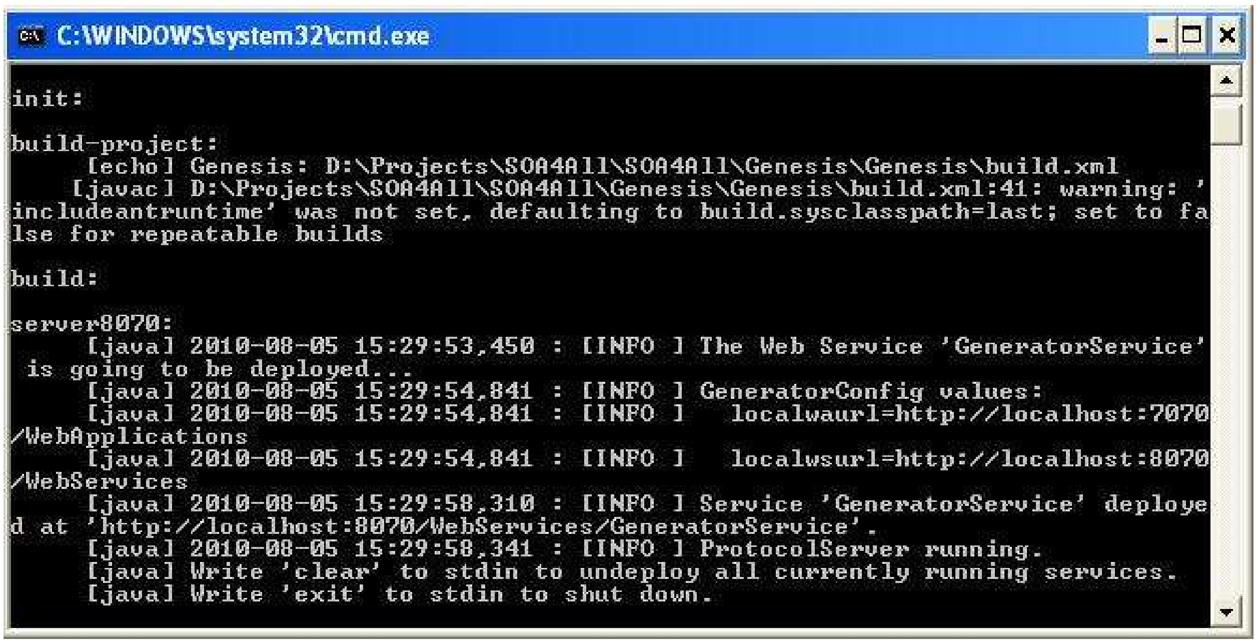
- for a host with the address:
    - http://localhost:**8060**/WebServices/GeneratorService

  the following script should be executed: **ant8060.bat**
- for a host with the address:
    - http://localhost:**8070**/WebServices/GeneratorService

  the following script should be executed: **ant8070.bat**
- for a host with the address:
    - http://localhost:**8080**/WebServices/GeneratorService

  the following script should be executed: **ant8080.bat**

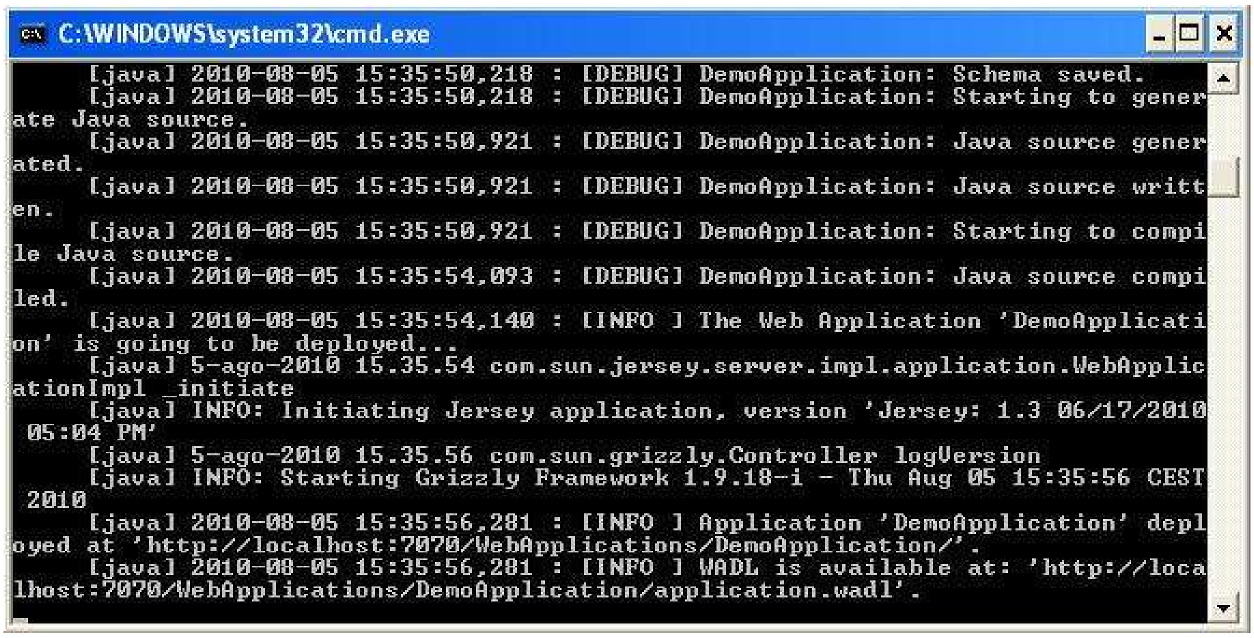
For each of the script files that has been launched, one has to wait until the shell shows something like this:



```
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 fa
lse for repeatable builds

build:

server8070:
     [java] 2010-08-05 15:29:53,450 : [INFO ] The Web Service 'GeneratorService'
 is going to be deployed...
     [java] 2010-08-05 15:29:54,841 : [INFO ] GeneratorConfig values:
     [java] 2010-08-05 15:29:54,841 : [INFO ]   localwaurl=http://localhost:7070
/WebApplications
     [java] 2010-08-05 15:29:54,841 : [INFO ]   localwsurl=http://localhost:8070
/WebServices
     [java] 2010-08-05 15:29:58,310 : [INFO ] Service 'GeneratorService' deploye
d at 'http://localhost:8070/WebServices/GeneratorService'.
     [java] 2010-08-05 15:29:58,341 : [INFO ] ProtocolServer running.
     [java] Write 'clear' to stdin to undeploy all currently running services.
     [java] Write 'exit' to stdin to shut down.
```

Figure 5: Execution of .bat files

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



```
C:\WINDOWS\system32\cmd.exe
     [java] 2010-08-05 15:35:50,218 : [DEBUG] DemoApplication: Schema saved.
     [java] 2010-08-05 15:35:50,218 : [DEBUG] DemoApplication: Starting to gener
ate Java source.
     [java] 2010-08-05 15:35:50,921 : [DEBUG] DemoApplication: Java source gener
ated.
     [java] 2010-08-05 15:35:50,921 : [DEBUG] DemoApplication: Java source writt
en.
     [java] 2010-08-05 15:35:50,921 : [DEBUG] DemoApplication: Starting to compi
le Java source.
     [java] 2010-08-05 15:35:54,093 : [DEBUG] DemoApplication: Java source compi
led.
     [java] 2010-08-05 15:35:54,140 : [INFO ] The Web Application 'DemoApplicati
on' is going to be deployed...
     [java] 5-ago-2010 15.35.54 com.sun.jersey.server.impl.application.WebApplic
ationImpl _initiate
     [java] INFO: Initiating Jersey application, version 'Jersey: 1.3 06/17/2010
 05:04 PM'
     [java] 5-ago-2010 15.35.56 com.sun.grizzly.Controller logVersion
     [java] INFO: Starting Grizzly Framework 1.9.18-i - Thu Aug 05 15:35:56 CEST
 2010
     [java] 2010-08-05 15:35:56,281 : [INFO ] Application 'DemoApplication' depl
oyed at 'http://localhost:7070/WebApplications/DemoApplication/'.
     [java] 2010-08-05 15:35:56,281 : [INFO ] WADL is available at: 'http://loca
lhost:7070/WebApplications/DemoApplication/application.wadl'.
```

Figure 6: Final deployment

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

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

3. Test Plan and Validation Processes

In this section we summarize a consolidated test plan and outline the validation processes that were undertaken for the different parts of the SOA4All runtime infrastructure. The SOA4All runtime includes the service bus and semantic spaces implementation of WP1, the service location components, mainly discovery, from WP5 and the components that make up the service construction suite from WP6.

The distributed service bus and the spaces provide a baseline communication and coordination infrastructure for Internet-wide service computing, and hence properties such as distribution, decentralization and performance are the determining factors when validating the project outcomes. Essentially, these core infrastructural services have to be evaluated in terms of scalability under increasing network size (number of nodes), changing network heterogeneity – i.e., cross-domain communication, aka inter-service park communication – and increasing service invocation load. To this end, the semantic descriptions have no relevance when testing the SOA4All infrastructure from WP1.

While for the service bus evaluation setting the number and physical location of service endpoints is important, in regards to service location (WP5), the focus is on the number of service annotations and the complexity of service descriptions; discovery does neither care about the location of a service nor about how it is implemented. Distribution, and hence the number of service discovery engines is not the primary target of our evaluation, although, experiments in decentralized and cloud-based settings complete the picture. Decisive factors are the query answering times and the precision and recall under changing and increasing numbers of service descriptions taken into account.

The task of service discovery is to find a matching service via its descriptions out of a collection of several thousands of service annotations. In service construction, the number of services taken into account is linearly dependent on the number of actions in a process; and hence, for all our use cases and scenarios is limited to a low countable number. Rather than by scalability, the evaluation of WP6 results is driven by suitability and applicability tests. In other words, service location components are responsible for reducing the number of relevant service descriptions to a small enough number so that the service construction suite can provide optimal service to process modelers from an initial goal to a final deployment on the distributed service bus and the invocation via service bus.

3.1 SOA4All Runtime Infrastructure

The WP1 Test Plan consists of two separate parts, in order to evaluate the two main results of this work package. The first section discusses the various tests and experiments to evaluate the fDSB, while the second section summarizes the tests for the Semantic Space.

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 infrastructure developed in T1.5. 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. Finally, the service generator GENESIS was used to provide a realistic amount of test services for additional performance and scalability tests.

3.1.1.1 fDSB Deployment and Integration Tests

For testing the fDSB deployment and integration, we carried out a large deployment of more than 700 DSB nodes over the testbed resources. In order to obtain a realistic environment for this deployment scenario, 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 table below shows the number of deployed nodes in our tests.

	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

3.1.1.2 fDSB Communication and Scalability Tests

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.

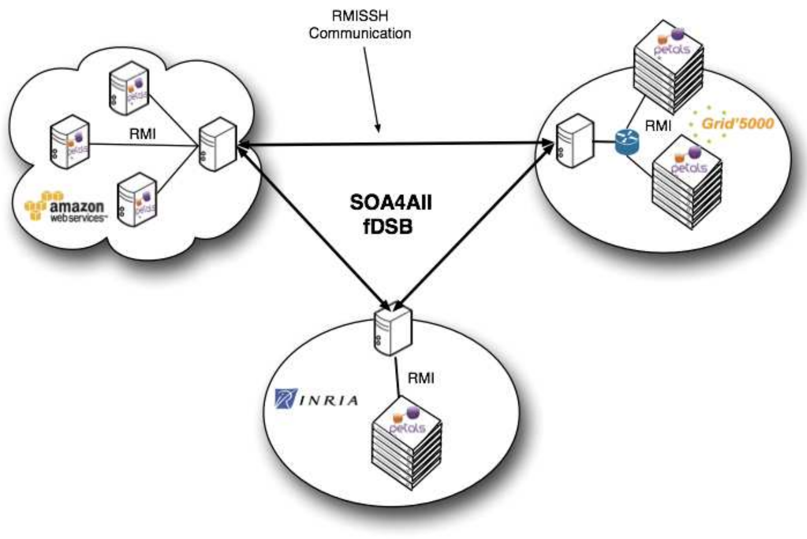


Figure 7: Testbed deployment

The service invocation experiments make use of a popular service benchmarking tool called SOAPUI. 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. Performance results of such invocations were collected and analyzed to verify the expected overhead of the fDSB layer.

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.

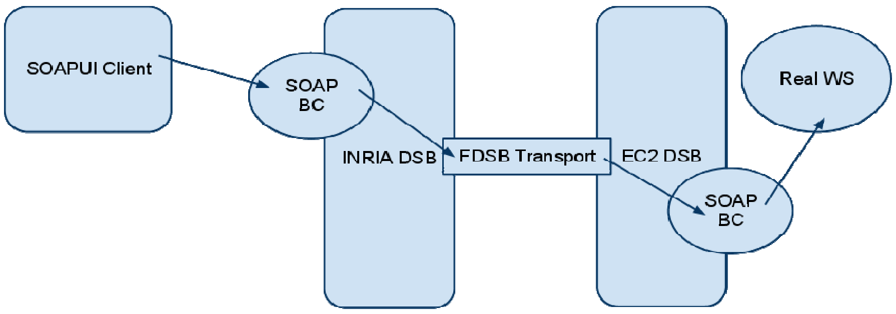


Figure 8: fDSB Service Invocation Path

In addition to single-client experiments, we carried out experiments involving multiple clients. These experiments evaluate the performance of the fDSB under different load conditions, which is obtained by the increment on the number of clients and number of threads by client.

In Section 4.1, we present more details on the experiments and performance results of such invocations and analyze the result to verify the expected overhead of the fDSB layer.

3.1.2 Distributed Space Evaluation

3.1.2.1 Performance and Scalability Measurements

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.

3.1.2.2 Design-Time Configuration and Scalability-Minded Analysis

To avoid performance and scalability limitations at runtime, it is important that relevant functional and non-functional properties of semantic spaces, such as latency, availability, or completeness, are analyzed at design time. It is difficult to detect and correct scalability problems by system testing only [13] – especially when problems are caused by conceptual flaws. Considering user and infrastructure requirements/constraints, such as available processing cycles or the storage size, allows for modeling the design-time evaluation as a constraint satisfaction problem (CSP, [17]).

For semantic spaces, we developed a CSP-based scalability analysis tool taking as variables the non-functional properties of a space, as constraints user requirements, system constraints and trade-offs between functional and non-functional properties [16]. In fact, the analysis of potential space realizations was reduced to a particular configuration design problem referred to as parametric design. In parametric design, a configuration is determined by pre-defined parameter template that defines the solution in terms of variables, possible value assignments and the design space given by requirements and constraints over the parameters. The outcome of the design task is the set of instantiated parameters that satisfy the design requirements and constraints, and in our case, offers a suggestion for a fit-for-purpose space implementation plan.

More technically speaking, parametric design is a search problem. Searching in complex domains is a knowledge-intensive reasoning task that is addressed by means of problem-solving methods that, through sophisticated knowledge models, resolve global solutions, often based on locally optimized search results. Our tool is based on a generic approach to parametric design problem-solving presented in [17] that leverages four independent knowledge models, ontologies, referred to as TMDA: Task, Method, Domain, and Application. The task model represents the parametric design task; the method ontology models the problem solving artifacts, while the domain is semantic spaces and the application the analysis tool. Building on TMDA, the tool is implemented as Web application running over the Internet Reasoning Service IRS-III that is developed at Open University [13]. A Web form allows space architects to specify their functional and non-functional requirements and preferences, as well as infrastructure constraints, such as available machines, and resource-related characteristics such as the overall targeted capacity of a distributed space or the use of the space; i.e., publication-intensive or retrieval-dominated access.

The evaluation of the tool was conducted qualitatively via expert consultations. The selected experts are the architects of different applications that rely on semantic space technology for improving the flexibility and scalability of their implementations, or that have concrete plans to do so. While the applications were very different in their purpose, scope and non-functional requirements, they all share the need for an infrastructure for large volumes of semantic data. The four scenarios were the European Patient Summary, Life Science Data Integration, Multimedia Content Marketplace, and the SOA4All Service Bus Monitoring Platform.

We assessed the tool in terms of program evaluation, impact, and efficiency/ cost-benefit. Prior to the interviews, the expert studied the user manual to get familiar with the objective, the promised functionality and the conceptual models of the tool. After clarifying the baseline, the experts had the possibility to test and play with the tool for 15 to 30 minutes. While playing with the tool, the experts were assisted if required, and could ask questions about the tool and its implementation. The evaluation interviews were conducted immediately after the testing phase by means of 17 open-ended questions covering all three aspects of the evaluation. The experts highlighted the simplicity and the automated transformation of requirements into a realization plan. The quick access to an infrastructure proposal was considered to be very powerful. From a usage perspective, the tool is seen to be very easy to use, and well-assisted with limited parameters; there is no need for much training. The results of the impact evaluation showed that finding a proposal for a space realization by means of an analytic tool is much more cost effective, safer and of significant impact. The evaluators agreed that the benefit of the tool is significantly higher than the costs. This is not surprising, as the tool was deployed as Web application with the reasoning engine running on a public server hosted by a third party. Further details about the tool, its implementation and ontologies as well as the evaluation are to be found in [16].

3.2 SOA4All Service Location

Given that service descriptions crawled by seekda mainly represent the information derived from WSDL service descriptions, we decided to synthesize rich semantic service descriptions in a fairly large scale to perform our measurements. Therefore, for the evaluation of the (non-distributed) semantic service discovery approached developed during the project, we created a set of randomly generated service descriptions with varying size ranging from 5,000 to 30,000 descriptions, which is approximately the number of currently available Web service according to seekda[2]. We used the Semantic Web for Research Community (SWRC) ontology as domain knowledge to model service descriptions. It provides classes and properties to express individual types and conditions.

Query sizes tested in the experiment.

		Small	*Medium*	*Large*
Query size				
	Variables	*6*	*9*	*12*
	Relations	*9*	*12*	*15*
	NFRs	*2*	*4*	*6*

We measured the mean query answering time of the reasoner on a quad core Xeon CPU (2.33GHz) powered machine. Queries of three different sizes (small, medium, large) were sent to the reasoner. Small, medium, and large conjunctive queries with various numbers of inputs/outputs, and relationships among them within precondition and effects, were tested in this experiment. The Table above lists the precise number of terms of the individual queries. As depicted below, the time to answer these queries range from 2.8s, 4.2s, 5.0s with 5,000 service descriptions to 17s, 23s, 33s with 30,000 descriptions for small, medium, large sized queries, respectively.

[2] http://webservices.seekda.com/about/web_services

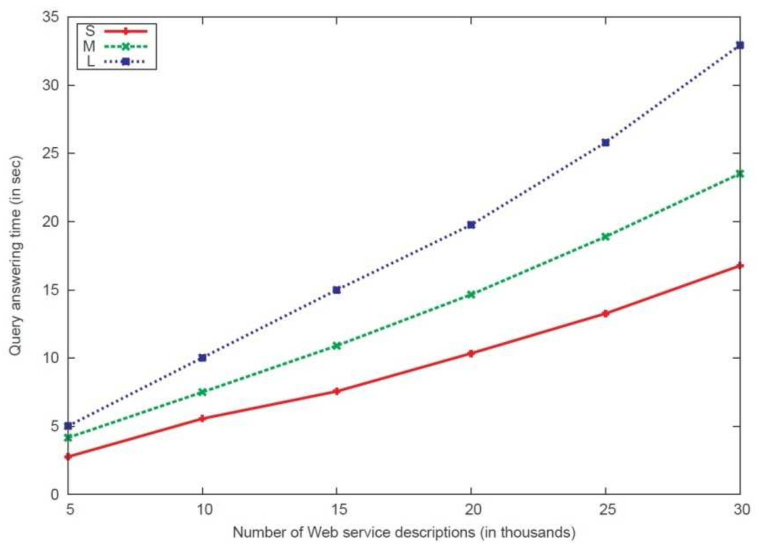


Figure 9: Mean query answering time against increasing number of Web service descriptions for three query sizes.

Note that the purpose of the figure above is to show the feasibility of the presented discovery approach. It is clear, that query answering time measure highly depends on size and structure of the used domain ontologies, size and complexity of the query and service descriptions. Nevertheless, these results can be significantly improved by the introduction of indexing structures, increasing the computational power, and distributing either the reasoning process or, independently, the processors carrying out discovery.

As sketched at M30 in Deliverable 5.4.2 (as well as D5.3.2), the integration of the discovery and ranking approaches in DisCloud, and specifically the creation of a repository for service templates (i.e., not just for service descriptions) has allowed a distributed approach to discovery and ranking to be investigated in the latter stages of the project. Evaluation of this approached has proceeded, as planned, based on the KIT component of the OpenCirrus cloud computing research testbed[3]. As planned, the implementation of this prototype has been carried out using Hadoop, though management of the cluster for testing and (scalability-oriented) evaluation has used the OpenNebula overlay (an open source cloud management toolkit, with significant contribution from EU-funded research) rather than the planned (commercial solution) Eucalyptus.

At the same time, work on Linked Open Services that has emerged from, and been supported by, SOA4All[4] as a Linked Data-oriented exploitation of project results, has motivated that the service models developed in the project be used in combination with the SAWSDL-compliant annotation scheme to link to data-centric service descriptions with input and output based on SPARQL graph patterns [13]. Motivated also by feedback from the project's tool user/functional evaluation, the distributed discovery/ranking approach has therefore concentrated on matches between service descriptions and templates using such graph patterns, and has accommodated a new notion of 'partial match', based both on predicate subsets in the graphs and on identified resource subsets.

The full results of the evaluation are communicated in the updated version of the Service Ranking Prototype in D5.4.3 (introduced in project Description of Work v19). In order to illustrate the approach taken to scalability-related evaluation, however, the details of the

[3] https://opencirrus.org/

[4] http://www.linkedopenservices.org/events/tutorials/ISWC2010/

updated (graph pattern based) service description and template generation algorithms are reproduced here.

Firstly a set of global parameters are set, as follows:

t	*The number of templates to be generated*
s	*The number of service descriptions to be generated*
p	*The number of predicates in the 'global pool' (there is no need to separate predicates into different named graphs, so these will simply be expanded as:* P = {*http://www.example.com/vocabulary#predicate1,* *http://www.example.com/vocabulary#predicate2,* *…,* *http://www.example.com/vocabulary#predicatep}, so that* $\|P\| = p$
r	*The number of identified resources in the 'global pool'. Again* R = {*http://www.example.com/vocabulary#resource1, …},* $\|R\| = r$
i_{min}, i_{max}, o_{min}, o_{max}	*Respectively the lower and upper bounds on the number of input triple patterns and output triple patterns per service description/template*
p_{min}, p_{max}	*Respectively the lower and upper bound on the number of predicates in each local pool,* P_j*, per service description/template, used in generation*
r_{min}, r_{max}	*Respectively the lower and upper bound on the number of resources in each local pool,* R_j*, per service description/template, used in generation*
v_{min}, v_{max}	*Respectively the lower and upper bound on the number of variables in each local pool,* V_j*, per service description/template, used in generation*
iv_s, iv_p, iv_o, ov_s, ov_p, ov_o	*The probability (*$0 <= iv_s, iv_p, iv_o, ov_s, ov_p, ov_o <= 1$*) that a variable will be used in the subject, predicate and object positions, respectively, for each input and output triple pattern*

Based on these parameters, as each service description/template, j, is generated, the following local parameters (i.e., applying only to that description or template) is generated:

p_j	*The number of predicates in the 'local pool' for the description/template* *(*$p_{min} <= p_j <= p_{max}$*), leading to randomly-selected* $P_j \subseteq P$, $\|P_j\| = p_j$
r_j	*The number of resource in the 'local pool' for the description/template, again:* *(*$r_{min} <= r_j <= r_{max}$*), leading to randomly-selected* $R_i \subseteq R$, $\|R_j\| = r_j$
v_j	*The number of variables in the 'local pool' for the description/template. Simply* $v_{min} <= v_j$ v_{max} *and* $V_j = \{?v1, ?v2, ..., ?v_{vj}\}$ *(Trivially, then* $\|V_j\| = v_j$*)*
i_j, o_j	*The number of input patterns and output patterns, respectively, for the description/template*

The algorithm to generate pairs of graph patterns, for inputs and outputs, based on these parameters in contained in D5.4.3. Values for iv_s, iv_p, iv_o, ov_s, ov_p, ov_o are fixed for all evaluations, based on experience with existing Linked Open Services. For the other parameters three consistent sets of values are chosen, representing: simple descriptions (low i_{max}, o_{max} and v_{max}) for limited domains (low p and r); simple descriptions for broad domains (high p and r); complicated descriptions (low i_{max}, o_{max} and v_{max}) for broad domains.

For each of these three groups the evaluation is carried out on a non-distributed 'cluster' (1 node), a small Hadoop cluster (3 nodes), a medium cluster (10 nodes) and a large cluster (100 nodes).

The results for each set of parameters, on each four cluster configurations, will be decomposed into the time taken to compute six metrics:

- The degree of subset matching of the set of predicates used in each template input graph versus each service description input graph;
- The degree of subset matching of the set of predicates used in each service description output graph versus each template output graph;
- The degree of subset matching of the set of identified resources in subject and object position used in each template input graph versus each service description input graph;
- The degree of subset matching of the set of identified resources in subject and object position used in each service description output graph versus each template output graph;
- The satisfaction (a binary judgment, based on an ASK query) between the input graph pattern of the service description by the skolemization of the template input pattern;
- The satisfaction (a binary judgment, based on an ASK query) between the output graph pattern of the service template by the skolemization of the service output pattern.

For a given template these six metrics can be used to decorate the service description dynamically as non-functional properties (as are the seekda monitoring results, as presented in D5.4.2), and included – with relative weightings defined according to the established preference model – in subsequent ranking of services. This also allows the scalability evaluation of the semantic ranking approach (for which there were previously too few services to test).

3.3 SOA4All Service Construction

The SOA4All WP6 Service Construction package outcomes that were obtained along with the project lifetime until the M30 milestone are evaluated in D6.5.4. This deliverable validates the results of the light-weight and adaptive composition techniques in terms of applicability and suitability as well as other criteria.

The SOA4All Construction Platform outcomes require to be assessed in order to ensure that the main objectives and requirements have been achieved and the principles observed. Deviations of the SOA4All Service Construction platform features from the objectives and requested requirements must be identified and measured, and corrective actions or improvements proposed in order to minimize the user experience and the successful exploitation of the SOA4All results.

Hence, this evaluation process aims to: a) define the evaluation scope (in the context of the general SOA4All evaluation approach), b) identify and define the evaluation criteria that it is use to assess the Service Construction, c) use these evaluation criteria to assess the most relevant technical features of the Service Construction from an holistic view, collecting and analyzing the evaluation results, d) provide a detail analysis and explanation of the evaluation results, proposing improvements for future work that overcome the non-positive aspects of the evaluation.

In the early stage of the project, the SOA4All usability evaluation work was defined in D2.5.1. This work defined the SOA4All evaluation process, jointly accomplished by some technical and the case studies work packages. Moreover, this document identified the main SOA4All objectives and their potential evaluators: end users, use cases and technology, and, based on those beneficiaries, the sort of possible evaluations, respectively: usability, fit-for-purpose and technical. In this context, the internal WP6 Service Construction evaluation should complement the aforementioned overall evaluation schema.

In this sense, WP6 evaluation is essentially a technical heuristic evaluation of the Service Construction features, attending to internal (to WP6) scientific and technical requirements, and other criteria explained in next paragraphs, based on the experience gained during the development of the tools and the use case scenarios. However, the process modelling assisted features, supported by Design Time Composer (DTC) and Optimizer have been also evaluated experimentally within the use case scenarios measuring their performance and behaviour with regard to the scale of the knowledge bases used. Since the use cases knowledge bases are small, additional experiments were run using programmatically created knowledge bases, containing service and process descriptions, and process models.

WP6 evaluation approach is also holistic, focusing on the most relevant global functionality offered by the Service Construction rather than on individual components separately. The Service Construction main functionalities evaluated are the following:

- the balance between light-weightness, expressivity, complexity, correctness and executability for process modelling, since this balance was one of the main driven criteria for the specification of the LPML.
- the easiness on process modelling, including assisted features provided by DTC since SOA4All Service Construction targets unskilled end-users.
- the modelling by knowledge intensive reusability, since this modelling approach is also a driven criteria for the specification of the LPML and the assisted modelling support for Template Generator and DTC.
- context-awareness process adaptation, since context was one of the four main SOA4All principles, and in particular a driven criteria for Service Construction
- process optimization, as one of the most relevant modelling assisted features.
- process deployment adaptation, autonomous capabilities for process execution and hybrid process execution support as the most important Execution Environment features.

The balance between light-weightness, expressivity, complexity, correctness and executability concerning LPML has been evaluated heuristically. In particular, we have compared LPML with most relevant modelling languages used in the SOA context: BPMN and BPEL, targeting typical end-users: IT experts and business analysts; results of this evaluation are presented in D6.5.4 LPML has been also evaluated in the context of a user evaluation survey, using the metrics as well are grouped into technical, individual, organisational, and economic metrics. LPML correctness (syntactic, semantic, uniqueness and canonical, exchangeable format, coherency of different layers) is successful evaluated using static analysis of the LPML meta-model. LPML completeness and expressiveness is also evaluated in terms of ontological completeness and a pattern-based analysis. The results of this evaluation are as well collected in D6.5.4, Table 5..

The adaptability and extensibility of the LPML is analysed statically according to the design science. In general the semantic annotation allow for adjusting and extending the LPML. In D6.5.4, Table 6 discusses the LPML adaptability and extensibility facing the public sector extensions.

The usability evaluation of LPML is strongly aligned with the simplicity and understandability evaluation. First, a heuristic evaluation is performed based on literature criteria addressing a static evaluation according to the design science. Afterwards, an observational analysis is described in terms of a user survey and workshops focussing on the evaluation of the design of the LPML. The heuristic usability evaluation is based on heuristics for usability engineering focusing on user interfaces. These usability heuristics are applied to the LPML. D6.5.4 Table

7 describes the static analysis of these heuristics.

The easiness on process modelling, including assisted features provided by DTC is evaluated heuristically using the machine and human based computation principle. The behaviour of DTC assisting features concerning scalability and efficiency (performance) criteria are evaluated heuristically and experimentally. Heuristically we discuss architecture choices to adapt a single DTC service to scale up with the size of the knowledge base of service and process descriptions. Experimentally, we analyse the performance of DTC methods processing the models proposed in the eGovernment use case scenario. In order to test the DTC methods with scales exceeding the knowledge bases of the use cases, we have prepared programmatically a randomly generated knowledge base of service and process descriptions where we control their number. KB sizes in our experiments range from 10^3 to $2 \cdot 10^4$ descriptions. Similarly, we have prepared a controlled number of input process models and templates whose number of tasks is also configurable. We have invoked activity-level DTC methods 50 times and process-level DTC methods 10 times. BindActivity and resolveActivity methods answer within [0, 10] seconds for KB sizes ranging [10^3, $2 \cdot 10^4$], response times compatible with Web applications. ResolveProcess method answers within [10, 45] seconds for the same KB sizes. The response in case of resolveProcess method strongly depends on the number of design models posted by DTC onto its blackboard, and on whether DTC found a complete design model with data flow generated (in this case the SLO service is also invoked). In case the solution space is traversed resolveProcess execution time can get extremely high (some minutes, depending on the solution space size). Guard conditions can be programmed in DTC in other to guarantee a reasonable processing time, relaxing the completeness of the returned solution: Future work to improve DTC will limit the number of posted design models or just returned the best-found model solution within a pre-determined processing time. Experimental details are given in D6.5.4, section 3.3.

The modelling by knowledge intensive reusability feature is supported by the Template Generator and DTC. This feature is evaluated heuristically with regards to the template-based composition, reusable, composability, distributed, openness, ontology based, centrality of mediation, usability, autonomous criterions. This evaluation criterion identifies the main limitations and drawbacks of this approach and proposes improvements for future work.

Context-awareness process adaptation is also heuristically evaluated. Despite of the fact context awareness was one of the four pillars for SOA4All, it has received little interest, and, indeed context-awareness process adaptation feature has been only partially implemented in DTC (design time) and the Execution Environment (runtime). We discuss the few available results for context-awareness process adaptation in some of the eGovernment scenarios; we remark the limitations of this approach and propose improvements as future work.

Process optimization evaluation has been driven along three main directions: i) scalability, ii) optimization performance and iii) quality of optimization. The latter criteria are the most relevant to evaluate the optimizer component and to run comparisons with existing approaches. Since optimization problems are NP-Hard, it is crucial to evaluate the behaviour of our component in the context of SOA4All i.e., thousands of services. In addition, this level of evaluation has been motivated by the analysis of state-of-the-art approaches that all consider performance analysis with a large amount of services. In D6.5.4 we analyse the performances of our approach by

- discussing the benefits of combining QoS and functional criteria;

- comparing the evolution of the composition quality's factors over the GA (Genetic Algorithms) generations by considering both static and dynamic constraint penalties;

- observing the evolution of the composition quality over the GA generations by varying the number of tasks and candidate services;

- studying the behaviour of our approach regarding large scale compositions;

- evaluating performance, after decoupling the GA and the (on-line) DL reasoning processes which are both required in our approach;

- comparing, GA with IP (Integer Programming)-based approaches; and

- focusing, on the performance of the GA process by comparing the convergence of our approach with the [7].

Section 3.6 in D6.5.4 provides all the details of the results of optimization evaluation.

In addition, the evaluation of process deployment adaptation, autonomous capabilities for process execution and hybrid process execution features supported by the Execution Environment has been reported in D6.5.4.

4. SOA4All 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.

4.1 fDSB Evaluation

As explained in Section 3.1.1, we carried out 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 and to test the scalability of the fDSB with a growing number of clients.

4.1.1 Single-Client Service Invocations

In these first experiments, we evaluate the global invocation times for single-client invocations in different scenarios. The main goal of these experiments is to evaluate the impact of the fDSB usage in comparison with scenarios without the usage of the fDSB. It is worthy of notice that only scenarios involving local invocations (i.e. client and server in the same domain) are possible, because Petals does not allow inter-domain 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 invocation 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 invocation 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 (between Grid5000 and Amazon EC2 that goes through Internet between France and US), the overhead was 107.5% in relation to local fDSB calls, which is not negligible but acceptable considering that service invocations happens 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

4.1.2 Multiple-Client Service Invocations

As explained in Section 3.1.1, in addition to single-client experiments, we carried out experiments involving multiple clients, in order to evaluate the performance of the fDSB under different load conditions. These experiments were performed In two different scenarios: the first one contains invocations in the context of the Grid5000 platform, with very low latency (avg. ping 0.16ms) and high bandwidth, due to the fact that client and servers are connected through a fast backbone; and the second scenario, involving a higher latency (avg. ping between 15 and 29ms during the experiments) and lower bandwidth (i.e. across the Internet).

By selecting these two kinds of scenarios, we believe to provide a comprehensive evaluation, as the main factors that contribute to invocation time in real systems are network performances and the performance of the underlying service bus. Besides, in addition to global invocation times, these results present partial-times for the main three steps necessary for an invocation over the federated bus:

1. Service invocation time: this is the average time the service implementation takes to reply back the response for the service invocation.
2. Global Petals time: this is the average time spent within Petals buses. Considering an invocation over the federation, this time comprises the time spend in the DSB the client is connected to and the time spent in the DSB the server is connected to.
3. fDSB time: this is the time spent in the context of the federation layer. Since Petals communication in intrinsically local, this time also encompasses the time spent over lower-performance networks (i.e. the Internet), which is considerable, especially for the high-latency scenarios.

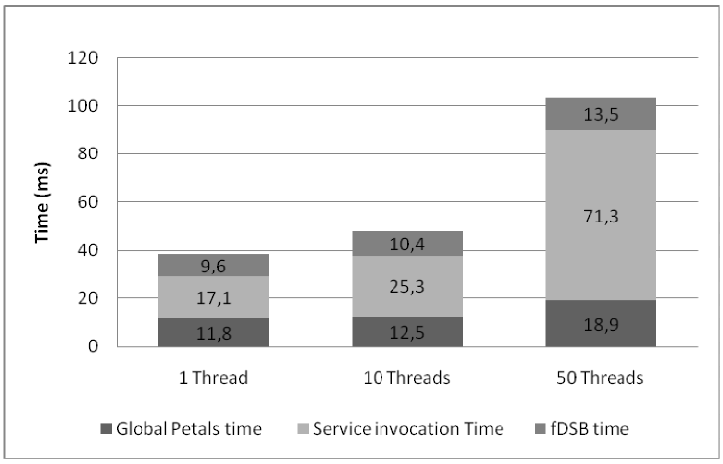


Figure 10: Low Latency fDSB Service Invocation (1 client)

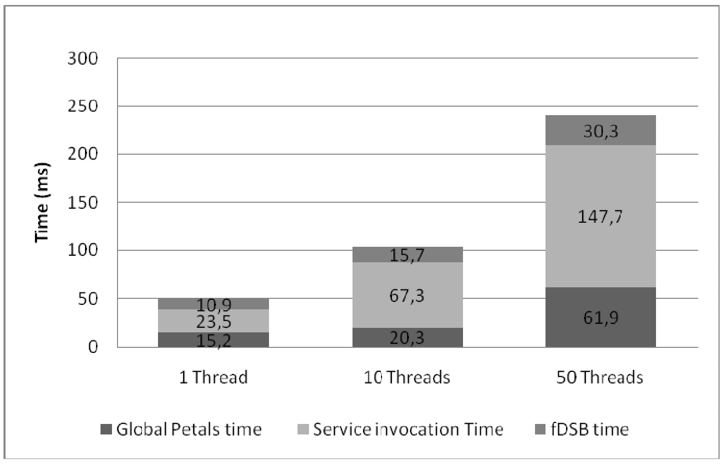


Figure 11: Low Latency fDSB Service Invocation (10 clients)

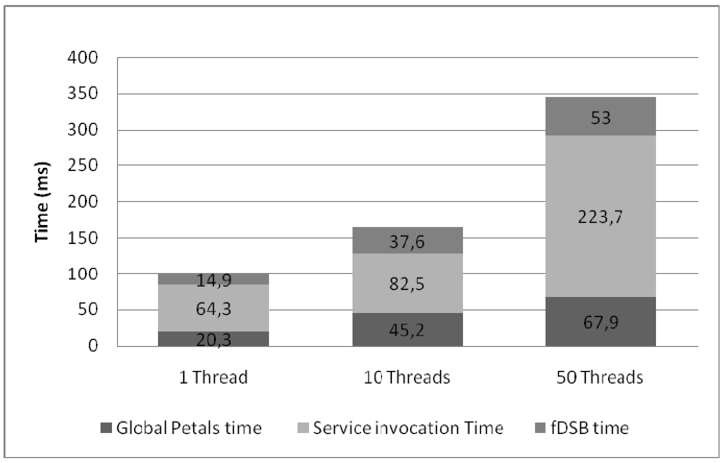


Figure 12: Low Latency fDSB Service Invocation (50 clients)

Figure 10, *Figure 11* and *Figure 12* show the invocation times across the fDSB in low latency conditions. A first important remark comes from the ratio between the different timers in the composition of the total time, where we can see that in all of the cases, the most expensive step (in terms of time) is the service invocation.

As expected, along with the increase of number of threads, we observe an increase of the average time spent in each of the three steps (Global Petals time, Service Invocation Time and fDSB time) in all of the three scenarios, which comes from the fact that resources used by clients and server are shared.

However, in all of the three scenarios, we can observe that the growth of the Service Invocation time (up to 628%) to be relatively higher than the growth of the other timers (355% for Petals and 407% for the fDSB). This means that the Petals and fDSB implementations scales slightly better than Apache Axis2 engine, the application server that hosts the service used for this experiments.

Besides, when comparing results across these three scenarios (in Figures 10, 11 and 12), we can see that the increase of time necessary to handle the different number of threads remained steady, despite of the number of clients. In any case, this growth can be consider small when considering that the number of simulated clients grew from 1 client / 1 thread) up 50 clients / 50 threads.

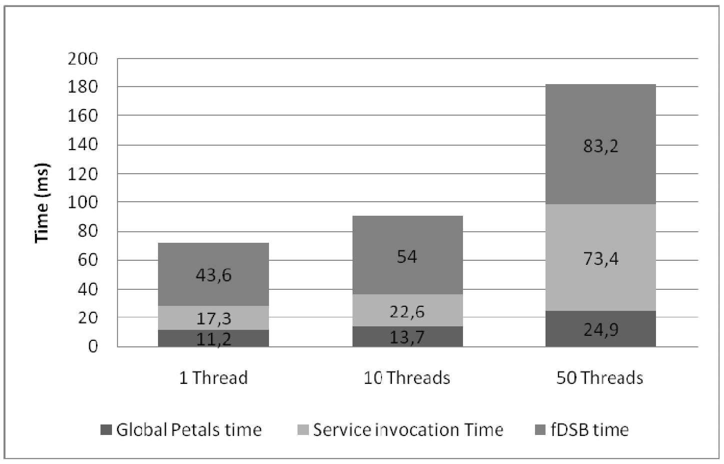


Figure 13: High Latency fDSB Service Invocation (1 client)

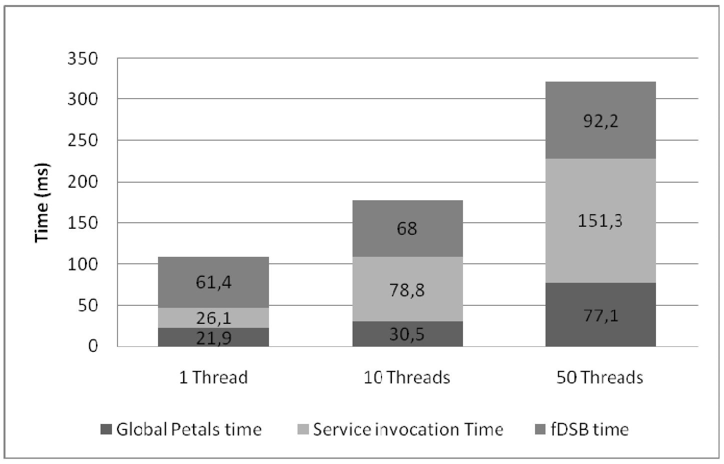


Figure 14: High Latency fDSB Service Invocation (10 clients)

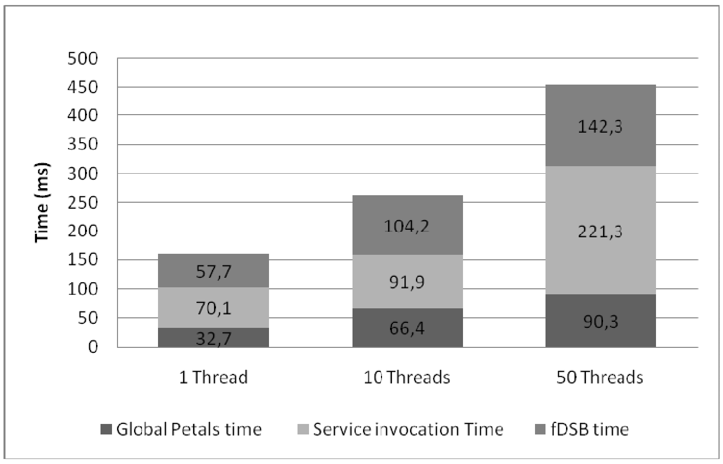


Figure 15: High Latency fDSB Service Invocation (50 clients)

Figure 13, Figure 14 and Figure 15 show the invocation times across the fDSB in high latency conditions. In a different way than the experiments in lower latency conditions, the fDSB time is more significative in these scenarios. This comes from the fact that, unlike Petals and Service Invocation times, which only require local communications, the fDSB is responsible for handling the inter-domain communications, which in these experiments influence in a bigger extent the overall communication time.

When comparing the growth in the fDSB time in the low-latency scenarios with the growth in the high-latency scenarios that the latency does not impact in the scalability of the fDSB. For instance, when comparing the execution with one thread, the fDSB time increases 172% between 1 and 50 clients for the low-latency experiments and only 132% for the high-latency experiments. However, we could not precisely assess the impact of latency as it presented important changes (between 15 and 29ms) during the experiments, probably due to the fact that physical resources are shared by Amazon EC2 instances and the performance depend on the number of running instances and their communication requirements.

By comparing the total time required for the service invocation between low and high-latency scenarios, we can notice that the impact of network latency does not increase with the number of clients. For instance, for 1 client and 1 thread, the increase was 173% and for 50 clients and 50 threads 134%. This result is interesting in the sense that the fDSB was created to support a federation of service buses, possibly geographically spread.

To sum up the fDSB performance evaluation, we believe that these results are very promising, in a sense that the fDSB did not presented a large overhead, even with relatively high load, up to the point that we could evaluate. Besides, Petals performance, which is equally important to the global fDSB performance, has also shown to scale well.

4.2 Semantic Spaces evaluation

4.2.1 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 16 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 17, 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.

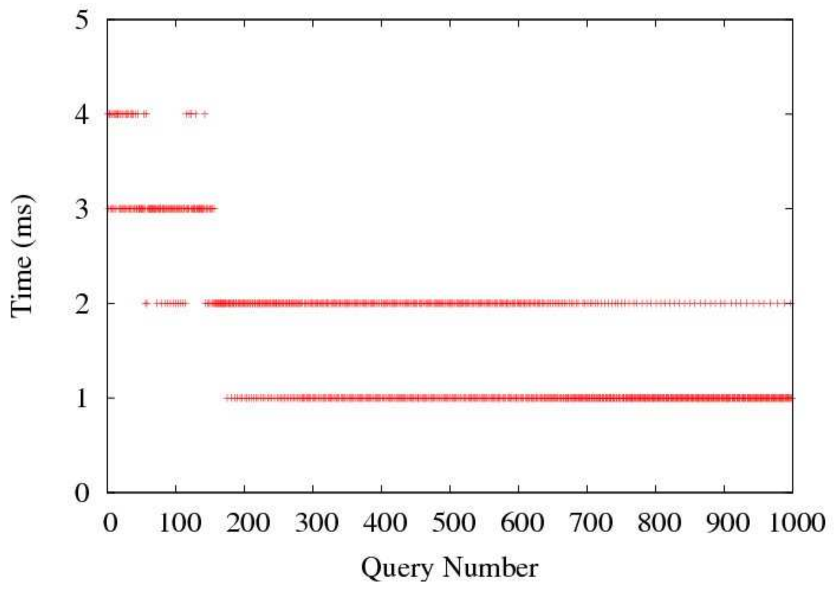


Figure 16: Individual time for sequential insertion of random statements on a single local peer

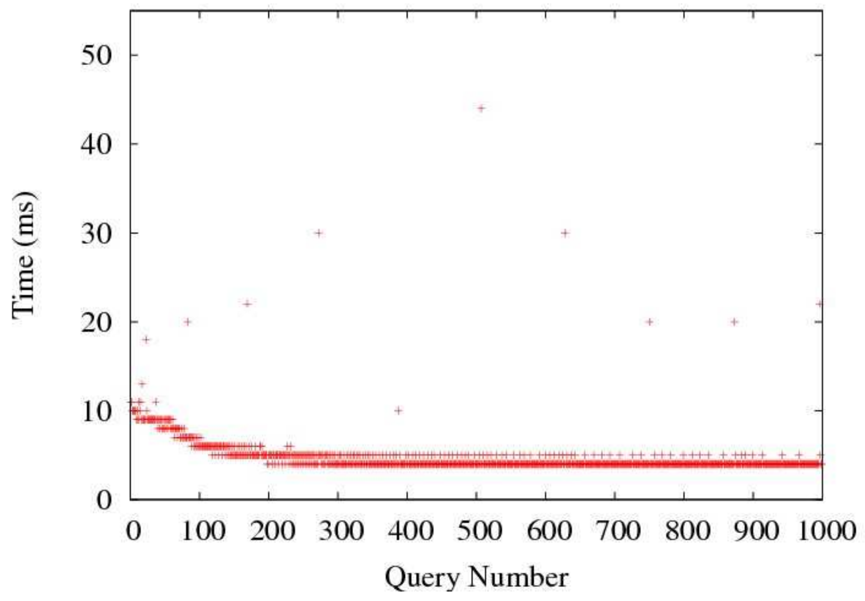


Figure 17: Individual time for sequential insertion of random statements on a remote peer

4.2.2 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 18 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 19 show a sharp drop of the total time, clearly highlighting the benefits of concurrent access.

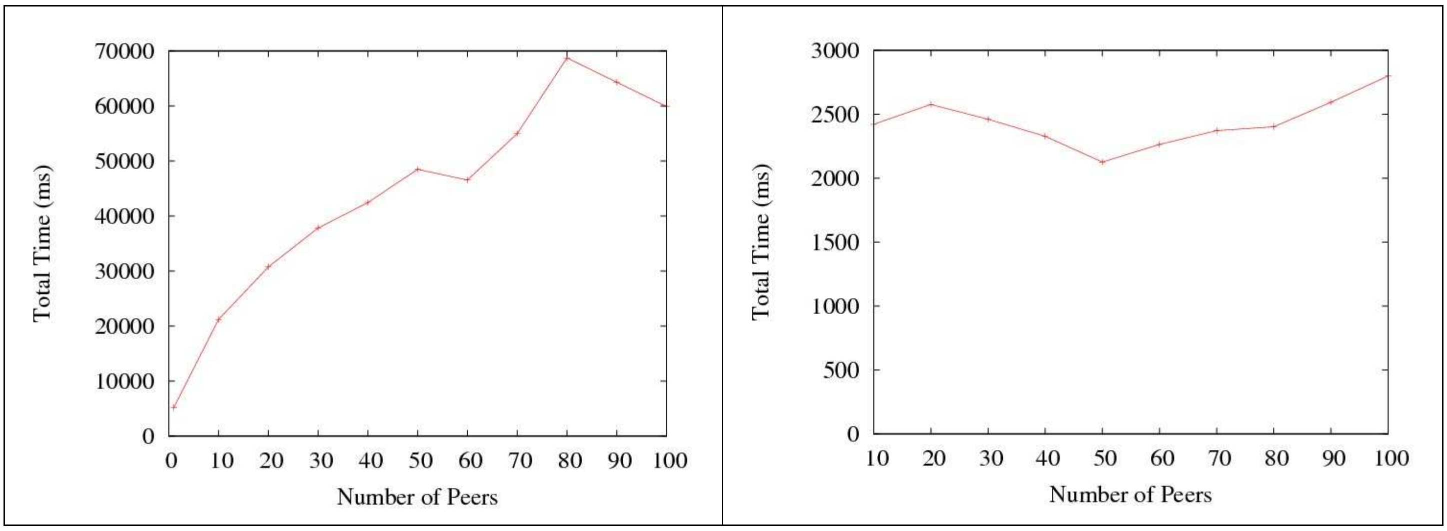


Figure 18: Insertion of 1000 statements for variable number of peers, 1 thread (left) and 32 threads (right)

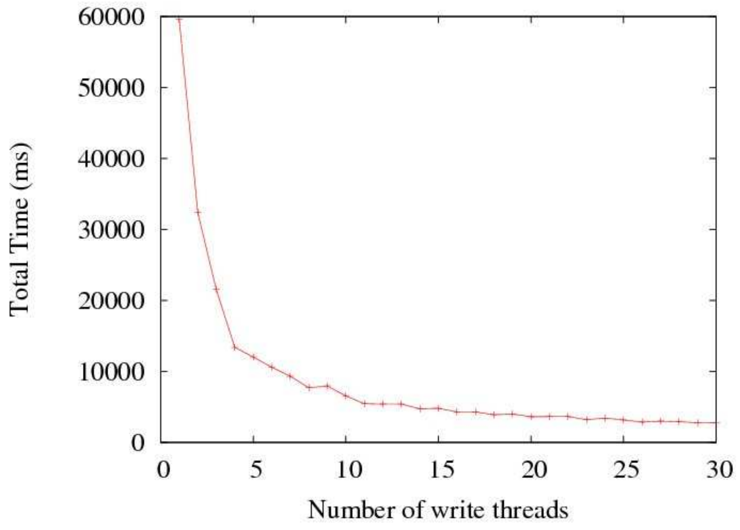


Figure 19: Evolution of the time for concurrent insertion on a 100 peers overlay

4.2.3 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 ecommerce 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://downlode.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 20 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.

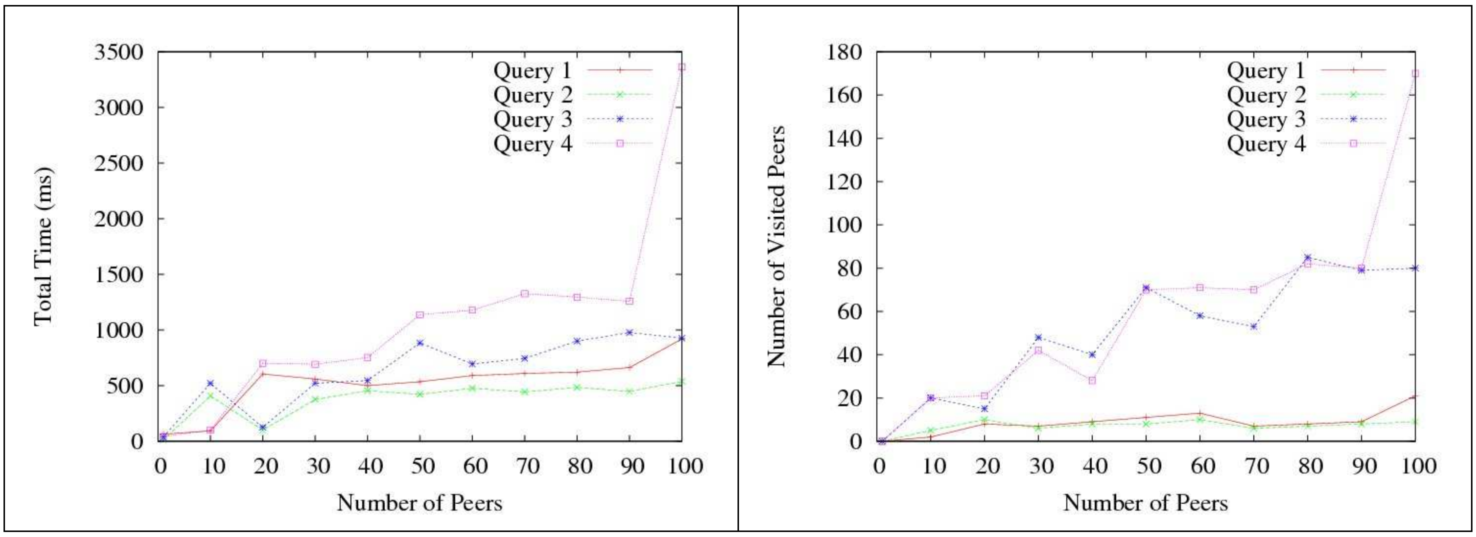


Figure 20: Custom queries with BSBM dataset on various overlays, execution time (left) and message overhead (right).

4.3 Comparison with other solutions

In this section we 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[5] that is an open source ESB. This ESB lets you to create internal enpoint refernces (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.

[5] Fast, open-source ESB, based on Apache Synapse, available at http://wso2.com/products/enterprise-service-bus/

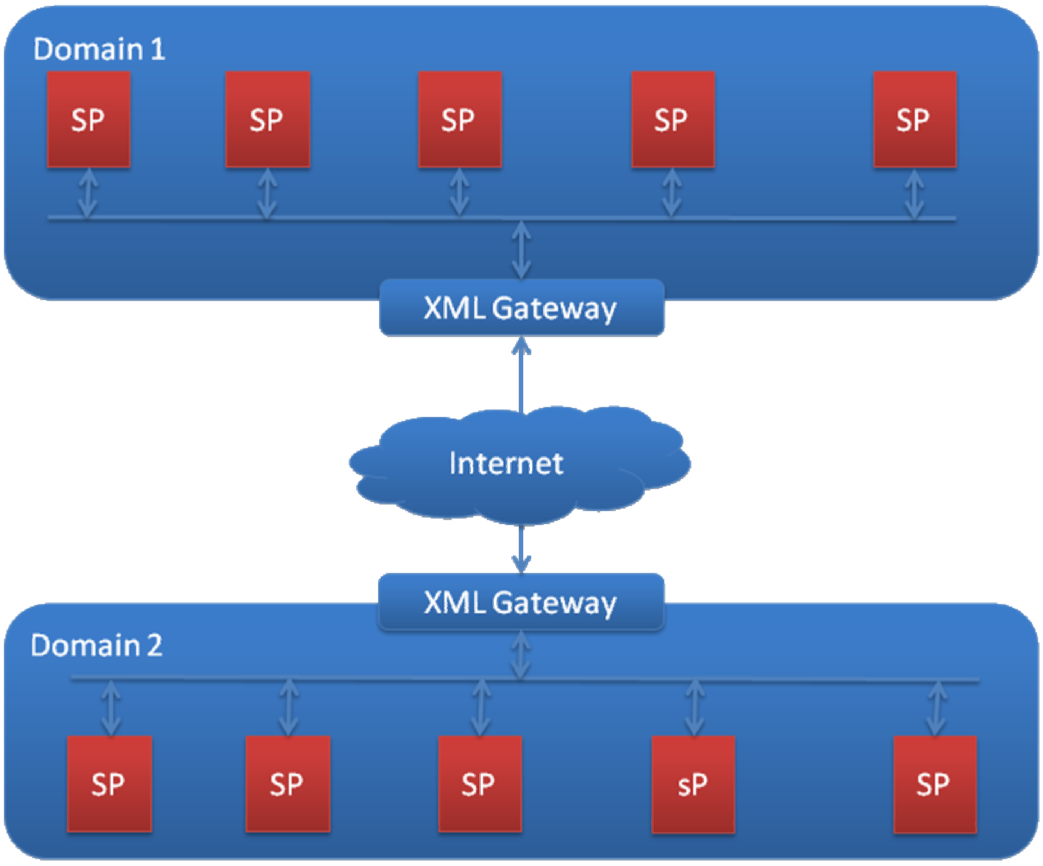


Figure 21 Gateways Scenario

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[6], Layer7[7], Cisco[8], Forum[9]).

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 while this is 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.

[6] http://www.vordel.com

[7] http://www.layer7tech.com

[8] http://www.cisco.com/en/US/products/ps6906/

[9] http://www.forumsys.com/products/xmlgateway.php

5. 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 the test plans and validation processes for the various components of the SOA4All runtime environment, including the fDSB, the semantic spaces, as well as the service location and service construction components.

Finally, this deliverable includes a report on the results of conducting the experiments based on these test plans. 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.

6. References

1. L. Juszczyk, 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. A.-L. Burness, R. Titmuss, C. Lebre, K. Brown, and A. Brookland: Scalability evaluation of a distributed agent system. Distributed Systems Engineering 6(4), 1999:129-134.
14. J. Domingue, L. Cabral, S. Galizia, V. Tanasescu, A. Gugliotta, B. Norton, C. Pedrinaci: IRS-III: A Broker-Based Approach to Semantic Web. Web Semantics: Science, Services and Agents on the World Wide Web 6(2), 2008:109-132.
15. R. Krummenacher, B. Norton and A. Marte: Towards Linked Open Services and Processes. Proc. of the Future Internet Symposium, 2010.
16. R. Krummenacher: A Parametric Design Approach to Scalability-Minded Management of Semantic Middleware. PhD Thesis, University of Innsbruck, 2010.
17. A. Mackworth: Constraint Satisfaction, in Encyclopedia of Artificial Intelligence, John Wiley & Sons Ltd, 1992: 285-293.
18. E. Motta: Reusable Components for Knowledge Modelling, Frontiers in Artificial Intelligence and Applications, IOS Press, 1999.

19. G.Canfora, M.Di Penta, R.Esposito, and M.L.Villani. An approach for qos-aware service composition based on genetic algorithms. In Proceeding of Genetic and Evolutionary Computation Conference, pages 1069–1075, 2005.
20. Y. Gorroñogoitia, F. Lecué, P. Un, G. Ripa, G. De Matteo. Evaluation of Service Construction. SOA4All report. 2011.

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.