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

D1.5.2 Setup SOA4All Testbeds

Activity N:	Activity 1 – Fundamental and Integration Activities
Work Package:	WP1 – SOA4All Runtime
Due Date:	M18
Submission Date:	14/09/2009
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	Hanival
Revision:	1.0
Author(s):	Bernhard Schreder HANIVAL Reto Krummenacher UIBK Sven Abels TIE Tomás Pariente ATOS Marc Richardson BT Matteo Villa TXT Giovanni Di Matteo TXT
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**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	15/07/09	ToC	Bernhard Schreder (Hanival)
0.2	27/07/09	Section 5 added	Sven Abels (TIE)
0.3	27/07/09	Added section on metrics, updates to service templates	Bernhard Schreder (Hanival)
0.4	28/07/09	Updates to all sections	Bernhard Schreder (Hanival)
0.5	30/07/09	Updates to Sections 2 and 3	Bernhard Schreder (Hanival)
0.6	05/08/09	Section 6 added	Reto Krummenacher (UIBK)
0.7	08/08/09	Updates to Sections 2 and 3	Bernhard Schreder (Hanival)
0.8	13/08/09	Updates to Section 6	Reto Krummenacher (UIBK)
0.9	17/08/09	Updates to Sections 2 and 3	Bernhard Schreder (Hanival), Reto Krummenacher (UIBK)
0.10	20/08/09	Updates to all sections	All
0.10.1	26/08/09	Review	Alex Simov (Ontotext)
0.10.2	26/08/09	Review	Maurilio Zuccalà (CEFRIEL)
1.0	01/09/09	Integration of review comments, Final version for submission	Bernhard Schreder (Hanival)
1.1	14/09/09	Final editing	Malena Donato (ATOS)

Table of Contents

List of Figures

List of Tables

List of Listings

Glossary of Acronyms

Acronym	Definition
API	Application Programming Interface
D	Deliverable
DSB	Distributed Service Bus
EC	European Commission
ES	Enterprise Service
EU	European Union
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
OSS	Operations Support System
REST	Representational State Transfer
SOA	Service Oriented Architecture
SUT	System under Test
URI	Uniform Resource Identifier
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 thus describes the different activities to realise a testbed environment and is separated in four main sections.

First, the overall objectives of evaluation and testing are summarized, and guidelines as well as metrics for the functional testing and the performance and scalability evaluation of the project results are proposed. The remainder of the deliverable then focuses on the three main enablers to facilitate the testing and evaluation activities.

The testbed infrastructure 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.

The second major part of the testing facilities describes the general build environment for SOA4All, based on the continuous integration tool Hudson. The testbed infrastructure is connected to this build environment via the definition of concrete build targets for the setup of testbeds and ensuing integration tests.

Finally, 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 discussion of different possibilities for a deployment plan and summarises the efforts to align the testing and evaluation tasks with other projects, in order to achieve the necessary scope to evaluate the scalability and performance of the SOA4All runtime.

1. Introduction

This deliverable describes the continuation of the work with Task 1.5, the SOA4All Testbed infrastructure and evaluation of project results. According to the work done and described in deliverable D1.5.1, a tool for the generation of testbeds has been selected and deployed to the SOA4All build and test environment. This deliverable now continues to describe the deployment of the testbed environment, the configuration of concrete testbeds, which are aligned to the SOA4All Use Cases and finally other tasks within the scope of T1.5 – including the deployment of the SOA4All runtime and the automated build system.

The ongoing work on extending the testbed environment is described as well, and – as a preparation for the validation and evaluation efforts to be conducted by the individual work packages – a guideline of applicable metrics and testing goals is provided.

1.1 Purpose and Scope

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

The deliverable defines a set of objectives and metrics for the various forms of evaluation within the scope of the project. It concentrates on the functional testing and the evaluation of performance and scalability characteristics of the project results, as other kinds of testing (e.g., concerning the user experience and interfaces) is discussed within workpackage 2.

The testbed infrastructure 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], [12] and [2]).

The second major section describes the general build environment for SOA4All, based on the continuous integration tool Hudson[1]. The testbed infrastructure is connected to this build environment via the definition of concrete build targets for the setup of testbeds and ensuing integration tests.

Finally, 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 discussion of different possibilities for a deployment plan and summarises the efforts to align the testing and evaluation tasks with other projects, in order to achieve the necessary scope to evaluate the scalability and performance of the SOA4All runtime.

1.2 Structure of the document

This document is structured as follows: Following this introductory section, Section 2 of this document continues with an observation of testing metrics and the overall goals of evaluation within the scope of Task 1.5, which defines a framework for the actual testing to be done by the individual work packages of the project. We provide an overview of the different kinds of tests and evaluation activities to be conducted within the scope of the project. Several objectives and metrics are discussed, and the role of the tools and activities provided and performed by task T1.5 are highlighted.

Section 3 then provides an update to the requirements for the testbed infrastructure, based

[1] Available at https://hudson.dev.java.net/

on the targeted alignment with the Use Cases. The services identified for the scenarios for each SOA4All Use Cases are used for the creation of testbed configurations later on. Following this update, the section then discusses the actual setup of the testbed environment, including the state of the plug-ins under development and the testbed configurations created.

Section 4 then continues by explaining the overall build system in use for SOA4All development and testing, including the continuous integration tooling in use and the links to the test targets. Finally, Section 5 concentrates on another aspect of the project evaluation – the test environment for the SOA4All runtime. The section explains where DSB nodes are deployed and mentions links to other projects SOA4All is collaborating with to realise a realistic amount of deployments with the project. The document concludes with an outlook on the evaluation tasks starting after M18, which will be summarised in the final deliverable within Task 1.5.

1.3 Alignment to SOA4All Evaluation

The testbed infrastructure specified in this deliverable will be used to evaluate the main objectives of the project from a technical perspective. The main roadmap for evaluation will be summarised as part of deliverable D2.5.1, and includes a set of metrics and performance indicators for the technical evaluation. Results from the evaluation process concerning these indicators will be reported in deliverable D1.5.3, which collects evaluation results from the experiments conducted with the testbeds generated by the testbed infrastructure.

2. Testing Metrics and Goals

Testing any kind of system or application makes no sense without the formulation of distinct objectives and goals, as well as metrics, i.e., quantitative measures, which establish the different scales along which to evaluate the SUT (System under Test). This section will formulate a set of metrics and goals for the evaluation of both individual services, SOA4All platform services and the overall SOA4All runtime.

These testing objectives focus on the technical evaluation of the project results. Further testing and validation of project outcomes is conducted according to the overall SOA4All evaluation plan, which was delivered as part of Deliverable D2.5.1, the Formative Evaluation and User-Centred Design [5]. This plan contains further information on the evaluation of – among other things – user interfaces, user experience and other aspects.

The remainder of this section is divided into two parts, with each part focusing on one specific area of testing, first by defining the overall objective and goals of the task. Furthermore, each part will explain which systems and components are targeted by the tests using these goals, and finally, the formal metrics are given where applicable. The first part discusses diverse methods of functional testing of software artefacts, while the second part concentrates on the evaluation aspects of one of the main project objectives – enabling a scalable platform, which is ready to deal with potentially billions of services.

2.1 Functional Testing

Functional testing is one of the core steps in any software development process and focuses on testing software artefacts based on their functional requirements. The functional testing should ensure that the program physically works the way it was intended and all required features are present. Furthermore, it should also ensure that the program conforms to the industry standards relevant to that environment.

In SOA4All functional testing is seen as an integral part of the development process by each team working on a platform service, the SOA4All runtime and the diverse tools developed in the project. The overall build system has been developed according to the requirements stated above, and is explained in more detail in Section 5 of this deliverable.

Further metrics and performance indicators for other aspects of SOA4All, such as non-functional properties, can be found in Deliverable D2.5.1. Examples for these aspects include increased robustness concerning service availability or the completeness and consistency of semantic annotations.

2.1.1 Metrics & Validation Methods

Concrete metrics for platform components will be formalised by the various technical work packages. As an example, Table 1 below lists the main metrics to be used for the technical evaluation of two different project results – the discovery component and the dynamic composition, respectively. The metrics are taken from a comprehensive table on evaluation metrics featured in deliverable D2.5.1. These evaluation objectives combine different aspects, including functional metrics, non-functional metrics and performance/scalability metrics. The owners of each technical component will need to define their own metrics along these lines. In the final deliverable of task T1.5, the testbeds evaluation (D1.5.3), all the different metrics, conducted tests and evaluation procedures will be presented, and the overall evaluation of the project results will be summarised.

Table 1: Technical Evaluation of the SOA4All components

Objective	**Metric**	**Definition**
Discovery	Non functional quality of retrieved services	This metric considers the non functional quality of services which have been retrieved (e.g., in terms of their Response time, Price, Reliability)
	Cover/Rest rate of each service	The cover rate considers the number of WS descriptions covered by the service and the query. The rest rate considers the number of WS descriptions required by the query but not provided by the service. The miss rate considers the number of WS descriptions provided by the service but useless for the query.
	Execution/Response Time and Scalability of the Discovery process	This metric signifies the time spent to discover services.
Dynamic composition	Non functional quality of Composition	This metric consider quality of services (i.e., QoS such as Response time, Price, Reliability) of each service involved in the composition.
	Semantic fit of composition	This metric consider the semantic quality of connections between services, This evaluates the data flow of any composition by considering their semantics.
	Execution Time of the composition process	This metric signifies the time spent to compose services.

As previously detailed in Section 5.1 of deliverable D1.5.1 [11], the main testing methodology for SOA4All means that test cases should be defined by each technology provider (including unit tests for specific functionalities of components or integration tests), but are also defined by dedicated testers, which conduct system tests and additional evaluation experiments (e.g. concerning scalability or performance of integrated components). These testers should be comprised of people, which are independent from the specific component owners. Thus, functional testing itself should comprise of several, separate steps, which are explained below in more detail.

- Unit Tests: Unit tests are used for the functional testing of specific functionalities on a fine-grained level, i.e. individual units of source code, such as classes. Unit testing is clearly within the responsibility of the different component owners and are included as part of the core development process. The main build environment, which has been set up for all the different (sub-) projects in SOA4All, has been provided to easily integrate unit testing with the build system. Section 4 provides more details on this environment.

- Component Tests: The functionalities of whole components, i.e. the verification of whether the components fulfil the specified requirements, should also be tested by the component owners. Testers will conduct similar tests, but also add additional experiments covering the aspects described above (i.e., specific evaluation objectives such as the semantic fit of service composition mentioned above). Again, the build environment described in Section 4 will provide the means to set-up such tests. In addition the generation of testbeds according to service templates (as detailed in Section 3), will be used to enable testing on this level.
- Integration Tests: Finally, the integration of the components via the SOA4All Studio and Runtime are going to be tested by the members of workpackages 1 and 2, as well as individual testers. Besides the objective of validating the functional requirements of the composed system, the main goals will be to evaluate the system under realistic conditions. In order to enable this kind of testing, T1.5 will provide a runtime test environment, by deploying the SOA4All runtime at a large scale. Section 5 of this document provides details on this deployment plan. Also, the generation of testbeds according to the means described in Section 3 will be useful to set-up realistic service environments.

In addition, the performance of different components and subsystems will be evaluated under stress-test conditions: Similar to the functional tests, stress-test conditions will be defined by the technical workpackages and component owners as the upper boundary of the presumed system size within which the technology is meant to function. For the SOA4All Runtime and Studio, an additional set of stress-test conditions will be defined, based on both the results of the individual components and the limitations imposed by the architecture, which are then used for integration tests whose objective is to evaluate performance and scalability of the overall runtime. The next section provides further insights on the nature and characteristics of these kind of tests.

2.2 Performance and Scalability Testing

Scalability is an indicator for performance changes in direct comparison to resource changes [6]; e.g., lower processing latency due to increased CPU power. It determines whether a system, network, or piece of infrastructure (e.g., Web server, database, or the service bus) has the ability to either increase workload in a graceful manner, to be readily enlarged to meet additional demand, or both, without replacement of hardware, and without the need for reengineering the system. If a system can be deployed in a wide range of configurations while maintaining an acceptable performance level under changing memory, bandwidth, users or data load, it is considered scalable. The challenge of scalability applies to SOA4All, as it does to any distributed systems.

Put in a more generic context, a scalable system must be economically deployable in a wide range of sizes and configurations [7], i.e., it should be re-applicable to various applications scenarios at lowest additional costs. Getting back to SOA4All, scalability is a property that indicates if the infrastructure is able to maintain an acceptable performance level by expanding in a graceful way when memory, bandwidth, operations, users and/or data are added.

This short introduction shows that scalability and performance testing are closely related. The objective for SOA4All will thus be in performance evaluations at different levels of scale, in order to evaluate the SOA4All Runtime and platform services also with respect to scalability. In Section 5 of this document, we present an outline of a deployment plan for large-scale installations of the SOA4All infrastructure. Although scalability discussions can already be made based on only a few nodes, we plan, per end of the project, to have deployment possibilities of up to one hundred nodes via the use of virtual machines or cloud infrastructures. Again, scalability in terms of users or services can be determined by generalizing measurements with changing loads in dimensions that are supported by a much smaller number of nodes.

Installing SOA4All on up to hundred nodes allows for testing the distribution behaviour and performance of the infrastructure under changing network sizes. Good performance of individual software components is a pre-requisite for a large-scale deployment, and certainly must be conducted on an individual bases. However, aiming for millions of users and billions of services requires solutions for scales that reach beyond the capabilities of single nodes or platform services. Therefore, distribution becomes very important for achieving the goals of SOA4All and for ensuring scalability. Large systems usually guarantee scalability through the partitioning of either the physical or the virtual world and thus by scaling out (adding more servers and distributing the work and data in the network) in addition to scaling up (increasing resources on an individual machine) [8]. Alternatively, by projecting this argumentation onto the SOA4All project, the former refers to the addition of further DSB nodes, while the latter refers to the optimization of individual software components of SOA4All and their hosting hardware.

Distribution has a particularly positive effect on the perceived performance in terms of responsiveness, as it is a core enabler of load balancing. Consequently, the frequency and number of requests that can be processed concurrently can be increased. Having the functionality of a system distributed across multiple machines allows to increase the overall throughput, which in turn sustains the required number of simultaneous users without augmenting the response times observed by them [9]. In the context of SOA4All, the performance is thus expected to change proportionally as additional infrastructural or platform services are added. In other words, an increase in utilization (i.e., a growing number of users and published data) must gradually be matched by a proportional growth in infrastructure without degrading the overall performance of the middleware.

In summary, besides the functional evaluation that was described in Section 2.1, we plan performance testing of individual components and scalability evaluations in the large. Distribution is an important factor in regards to scalability and we thus plan to deploy the SOA4All infrastructure in a network of up to hundred (virtual) bus nodes; compare Section 5.

2.2.1 Performance Model Parameters

For evaluating the performance, as a pre-requisite for discussing the scalability, we define a number of parameters for quantitatively describing the load, resources and performance indicators. These parameters provide the basis for first performance and scalability evaluations but will likely have to be adjusted in order to better meet the requirements and expectations of the SOA4All testing infrastructure. In that sense, the set of parameters given here provides the starting point for several iterations of testing that will conclude in month M36 with the presentation of the final evaluation results about the different platform services, the SOA4All Distributed Service Bus and the studio. These evaluations will be based on the final testbed that is delivered in month M30 with deliverable D1.5.3 on the testbed validation.

Load: The load parameters include numbers of services, process or users that are maintained and processed in the system. The listing gives indicative value ranges for the different load parameters. These ranges were determined according to the initial values given for similar parameters used for scalability evaluation in the TripCom project [14]. Note that not all combinations of load values are necessarily feasible (e.g., the combination of all maximal values reaches likely beyond the capability of the implementation, whereas setting only a subset of the values to the maximum would be tolerable).

- Number of Web services (invokable pieces of software) known by the infrastructure [10 … 1000]
- Number of Semantic Web services (semantic descriptions of service endpoints) registered and stored in the infrastructure [10 … 10000]
- Number of processes (semantically annotated compositions) maintained by the infrastructure [10 … 1000]
- Number of concurrent interactions with the infrastructure (users) [1 …1000]
- Number and frequency per second of invocations of individual platform services or the semantic spaces (a more controllable parameter than the number of concurrent interactions) [1 … 100], [0.01 … 10] for a resulting maximal load of 1000 accesses per second.

Resources: The resources parameters models the computational resources available to process or maintain the load indicated above.

- Number of bus nodes [1 … 100]
- Number of space nodes [1 … 100]
- Number of DSB nodes (hosting both the bus and the space logic) [1 … 100]
- Number of platform services of each kind [1 … 5]
- Number of concurrently used SOA4All Studio instances [1 … 10]

Performance: The performance parameters provide a means to quantify the system performance changes under different load and varying resource availabilities. The prime indicator for performance in the context of SOA4All is latency, or rather responsiveness.

- Response time, as the time interval between the invocation of a particular operation and the return of (successful) feedback.

The evolution of the responsiveness under changing load and changing resource availabilities eventually allows for assessing the scalability of the SOA4All implementation. Whether SOA4All or individual components are considered scalable or not depends on the expected performance progression of the investigated piece of software or protocol. The scalability of these is typically defined by complexity classes that are indicators for performance (e.g., O(n) for linear scalability). The target complexity class depends largely on the problem at hand. Search algorithms for example are generally considered to be scalable if they are in O(log(n)) -- e.g., binary search. Similar expectations, also O(log(n)), hold for response time and communication overhead in peer-to-peer systems – both in terms of number of nodes to visit (hops), and in terms of messages required to resolve a query. With respect to processing capabilities and memory, large-scale distributed systems have their target scalability mostly in the linear complexity class [10]: doubling the amount of memory should allow doubled as much data to be stored, increasing the number of servers should result in a proportional raise in the number of requests to be handled. Determining what scalability complexity classes can be considered ”good" or ”acceptable" is difficult to determine without having more detailed knowledge of the final system. As a rule of thumb, we expect linear scalability in terms of growing load in what concerns the evaluation of individual components. In regards to the overall infrastructure, sub-linear scalability is the targeted outcome of the implementation.

3. Setup of the Testbed Infrastructure

As explained in the previous sections, three different activities are ongoing within the scope of T1.5, in order to enable the various kinds of tests and experiments as detailed in Section 2. The first activity is a continuation of the efforts previously detailed in D1.5.1, and comprises of the setup of a testbed infrastructure. The objective of this infrastructure is to enable testers to generate (a realistic amount of) Web Services, respectively mocks of such services. In order to achieve such a generation, we utilize a testbed creation tool called GENESIS[2], previously described in Section 3.2 of D1.5.1 and in more details in [1]. We also provide service templates, based on the services identified by the various SOA4All Use Cases.

The following section first describes several changes and updates to the SOA4All Use Cases, which reflect the current state and available services of the Use Case storyboards as detailed in [3], [12] and [2]. The section presents the necessary updates to the alignment between the Use Cases and the testbed infrastructure.

3.1 Requirements Analysis Update

3.1.1 End-user Integrated Enterprise Service Delivery Platform

The End-user Integrated Enterprise Service Delivery Platform case study developed in WP7 has the public sector as its target domain [3]. This case study envisions an integrated service delivery platform based on the technologies and tools developed in SOA4All, allowing non-technical users in public administrations to handle typical administrative procedures. More specifically, using the Web-based tools of the SOA4All Studio, public servants of various governmental organizations will be able to search, model, annotate, modify, share, analyze, and execute administrative procedures in the form of lightweight business processes. These processes may be composed of SAP Enterprise Services, public Web services (hosted by 3rd party service providers), and human activities (to be executed by end users). Thus, the main result of WP7 will be an integrated demonstrator that addresses the specific needs of public administrations, such as the ones formulated by the EU Services Directive.

The SAP Enterprise Services offer complex business functionality like the management of resources or relationships with customers. Because of that, these services typically have large syntactic WSDL-based service interface descriptions that are difficult to understand for non-expert service consumers. Thus, by investigating how to make Enterprise Services consumable for non-experts, WP7 will significantly increase the number of services to be handled by SOA4All. Therefore, WP7 is in the process of developing a more simplified service layer that abstracts the complexity of these WSDL.

There are not extra requirements for the testbed infrastructure as the stated on deliverable D1.5.1 [11]. More details on available and planned services are described in deliverable D7.4. Below there is a list of useful links that helps to get more information and allows the discovery and consumption of the SAP Enterprise Services (ES):

- ES Workplace: http://esworkplace.sap.com
- Create login: Necessary to get and consume the SAP Enterprise services (https://www.sdn.sap.com/irj/sdn/soareg).

[2] Available at http://www.infosys.tuwien.ac.at/prototyp/Genesis/Genesis_index.html

- Overview ES Bundles: https://wiki.sdn.sap.com/wiki/display/ESpackages/Home/
- ES Workplace how-to: https://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/6240/
- ES Community: https://www.sdn.sap.com/irj/sdn/define-es
- Service Registry: http://sr.esworkplace.sap.com
- Manual: https://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/501668ab-976e-2a10-91b6-c1020e8c54f2/

The following table shows the services involved in the current version of the WP7 scenario:

Table 2: Test Web Services from the WP7 storyboard

Provider	Service Name	URL	Description
SAP	CustomerERPAddress BasicDataByName AndAddress QueryResponse_In	http://erp.esworkplace.sap.com/sap/bc/srt/xip/sap/ecc_customeraddressbasicdataqr?sap-client=800&wsdl=1.1&mode=sap_wsdl	Find the basic data of a customer using the customer's name or address (see ES Workplace)
SAP	CustomerERP CreateRequest Confirmation_In	http://erp.esworkplace.sap.com/sap/bc/srt/xip/sap/ecc_customercrtrc?sap-client=800&wsdl=1.1&mode=sap_wsdl	Create a new Customer in the SAP System (see ES Workplace)
SAP	CustomerBasicDataByID QueryResponse_In	http://erp.esworkplace.sap.com/sap/bc/srt/xip/sap/ecc_customer001qr?sap-client=800&wsdl=1.1&mode=sap_wsdl	Read Customer basic data (see ES Workplace)
SAP	CustomerERPBasicDataByID QueryResponse_In_V1	http://erp.esworkplace.sap.com/sap/bc/srt/xip/sap/ecc_custbasicdatabyidqr_v1?sap-client=800&wsdl=1.1&mode=sap_wsdl	Read Customer basic data (in Change Customer Bank Details context) (see ES Workplace)
SAP	CustomerERPBankDetailsByID QueryResponse_In	http://erp.esworkplace.sap.com/sap/bc/srt/xip/sap/ecc_customerbankdetailsidqr?sap-client=800&wsdl=1.1&mode=sap_wsdl	Read Customer Bank Detail (see ES Workplace)
SAP	CustomerERPBankDetailsUpdate RequestConfirmation_In	http://erp.esworkplace.sap.com/sap/bc/srt/xip/sap/ecc_custbankdetailsupdrc?sap-client=800&wsdl=1.1&mode=sap_wsdl	Update (create/change/delete) the bank detail of a customer (see ES Workplace)

WP7	Human Task Server	-	Management of human interaction with processes

3.1.2 W21C BT Infrastructure

BT's Web21c Use Case developed two separate scenarios for casual and business users, which are explained in detail in the D8.3 [12]. Currently, usage of Ribbit services requires a detailed technical knowledge of both Web service languages and programming languages, in particular Adobe Flex, JavaScript and/or PHP. Although documentation and guides have been provided, it is not straightforward to implement a composite service even for an advanced user, let alone a casual user, which Scenario 1 is aimed at. The aim of the case study is to provide semantically enhanced and expanded version of Ribbit, where the process of discovering, integrating, using and sharing Ribbit's services can be done much more effectively.

In the Scenario 1 (S1), focus is on the creation of simple mash-ups of BT services with other popular services available on the Web to create a new web application incorporating Ribbit services. The aim is to make it easy for novice users to get access to the facilities of the Ribbit services and combine them with other services on the Web. SOA4All will be used to overcome some of the current problems that limit the uptake of the Ribbit services, primarily the technical knowledge required and familiarity with programming languages such as PHP or JavaScript. As the focus of S1 is on casual users building non-critical applications, the scenario will involve minimal security or management infrastructure.

An example of the service composition that a S1 user would create is shown on the Figure 1 – a composition that allows user to organise a meet up with a group of friends in the last minute.

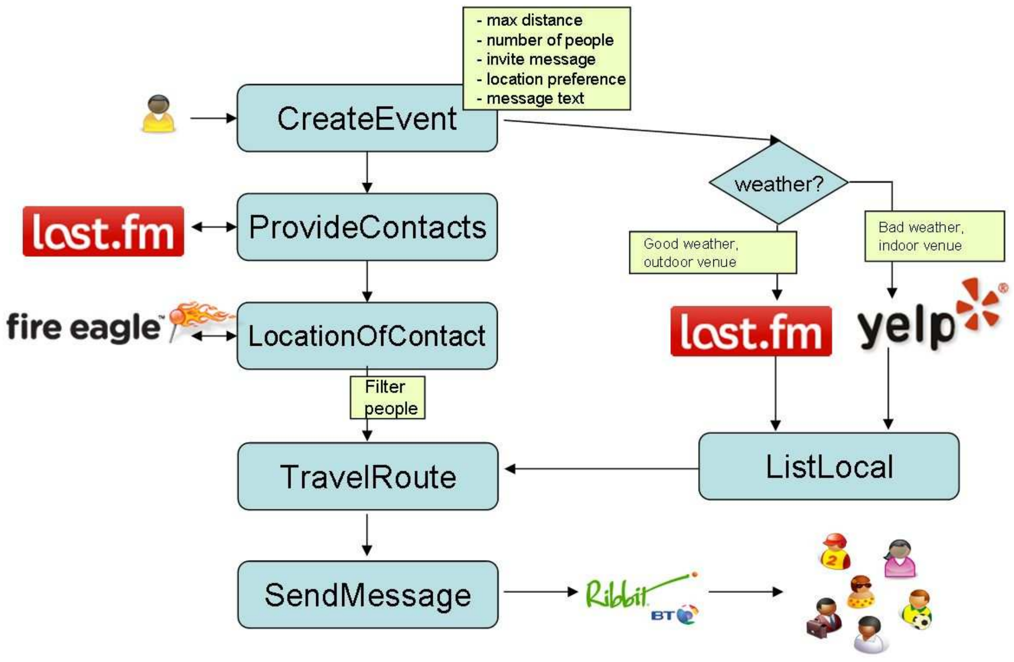


Figure 1 : Web21C BT Infrastructure Scenario 1

Scenario 2 (S2), details an "industrial strength" scenario and aims to utilise all the technical results of the project. In S2 businesses will use SOA4All technology to design and compose more complex end user applications to resell or use as part of their business, incorporating BT white label Ribbit services, their own services & Operations Support Systems (OSS) and some BT OSS services. This will enable creation of a business incorporating BT services, without complex face to face contract negotiations and manual work to integrate services, supporting businesses to go from 'idea to product' in minimal time.

An example service composition that a user would create in S2 is outlined in the Figure 2. In this example, a composition is created which allows a company to build a bulk text messaging service using the BT Ribbit SMS service.

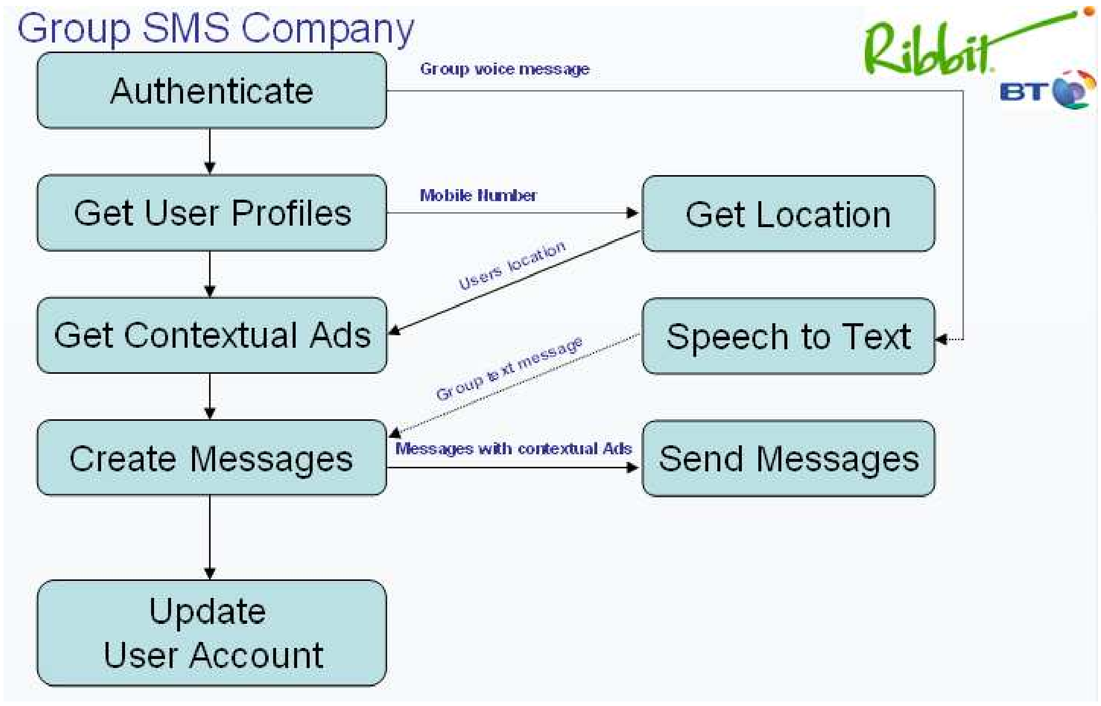


Figure 2 : Web21C BT Infrastructure Scenario 2

Since the case study is looking into integration of Ribbit web services with third party services already available on the Web, initial list of services has been created in the table in the Section 2.3.2 of the D1.5.1 [11]. Updated list of all planned services can be found in the D8.3 and the services used in the S1 are described in more detail in the D8.4 [13]. The following table shows a summary of services involved in the use case.

Table 3: Test Web Services from the WP8 storyboard

Provider	Service Name	URL	Description
Last.fm	ListEvents	http://www.lastfm.es/api/show?service=270	Operation name: geo.getEvents. Lists (predominately) music events and the venues for a given location (e.g. Concerts)

Yelp		http://www.yelp.com/developers/documentation/search_api	Retrieves list of different types of businesses and the reviews made of them
Fire Eagle	LocationOfContact	http://fireeagle.yahoo.net/developer/documentation/querying	Operation name: user. Retrieves the location of the user
Last.fm	ProvideContacts	http://www.lastfm.es/api/show?service=263	Operation name: user.getFriends. Retrieves the list of a user's friends
Multimap	ProvideRoute	http://www.multimap.com/openapidocs/1.2/web_service/ws_routing.htm	Operation name: Routing. Generates driving or walking directions between a set of locations
WeatherBug	ProvideWeather	http://weather.weatherbug.com/corporate/products/API/help.aspx	Operation name: getLiveWeather. Retrieves live weather data based on the location given
Ribbit	SendMessage	http://ngwr.labs.bt.com/Ribbit/myapp/RibbitSMSService.php	Sends a text message (SMS) to a specified number

3.1.3 C2C Service eCommerce

For the C2C eCommerce Use Case a new business scenario has been developed and is explained in detail in deliverable D9.2.1 [2]. In this scenario, users may combine various services and easily set-up eCommerce applications. The users are able to aggregate product data from third party suppliers, mediate between the different sources of product information and publish the products on their own web shop systems. Furthermore, the eCommerce framework enables them to address different syndication channels with their product data, submitting their offers to various social networking platforms, such as Facebook, Twitter, eBay or Google Wave. Figure 3 provides an overview of this process of collecting product data, and exposing the products to various external channels.

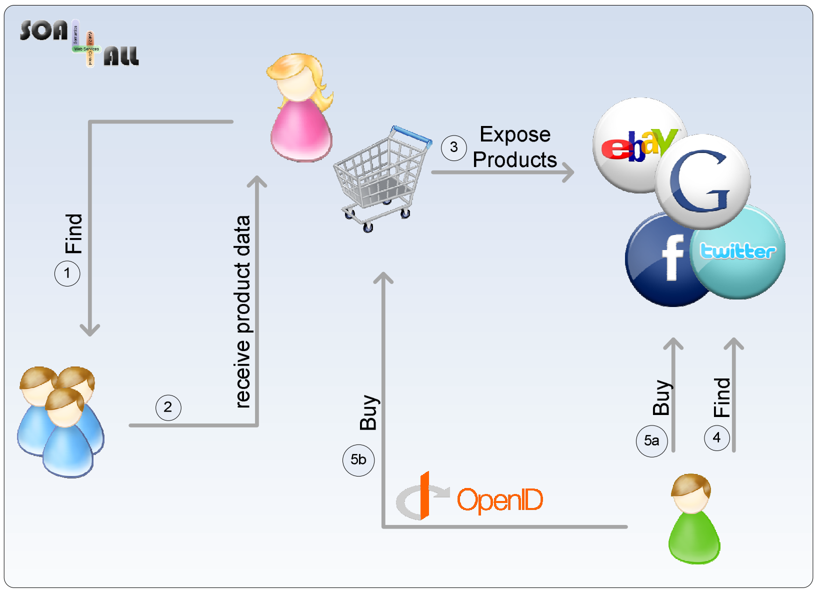


Figure 3: Updated C2C eCommerce scenario

Besides various process templates, goal templates and a customized User Interface for the SOA4All Studio, the WP9 eCommerce framework will offer a selection of services to realize this scenario, including:

1. Product services from different providers. For the WP9 scenario, several services will be realised, including several real world web services coming from existing webshops.
2. Support services provided by the WP9 eCommerce framework, which are necessary to complete the scenario (e.g. mediation services, or a collaborative advertising service)
3. Multi-channel export services, which enable the shop owner to automatically send new product data to several syndication channels, such as a dedicated Facebook application or a Twitter account.
4. Third party services provided by external parties in order to provide additional functionalities, like payment or credit rating checks.

Details on available and planned services are described in deliverable D9.2.1. The additional requirements for the testbed infrastructure, as detailed in Section 2.3.3 of deliverable D1.5.1 [11], are still valid for the updated WP9 scenario and have been taken into account for the development activities of the testbed infrastructure.

The following table shows a first collection of services involved in the updated version of the WP9 storyboard scenario. The services shown are realised for the first version of the WP9 prototype, due M24, and are therefore suitable candidates for sample testbed configurations. The actual configuration of a GENESIS testbed for these services is described in Section 4 of this document.

Table 4: Test Web Services from the WP9 storyboard

Provider	Service Name	URL	Description
TIE	ProductWebService	http://coconut.tie.nl:8181	This simple product service provides different operations to

		/services/productWS	show product lists, and query single products by either name or id.
TIE	MamboFiveService	http://coconut.tie.nl:8181 /services/mambooFive	A real world web service coming from an existing webshop, based on the MamboFive product. This service delivers real world data and is connected to a real data source.
Hanival	Chillydomains ProductService	http://hanival-products.at:9080/Hanival ProductWS	A web service hosted on Hanival's chillydomains ISP platform, which enables clients to get information on various products, their categories and single items.
WP9	MediationService	Not available yet	A mediation services which aggregates the product data from different suppliers and provides a product list without duplicates to the clients.
WP9	FacebookWrapper Service	Not available yet	A wrapper service for the Facebook application – this service stores the product data in a form which is reusable by the Facebook application from WP9.

3.2 Plug-In Development

In order to provide additional support for the requirements collected in D1.5.1 and in the previous section, several new plug-ins for the testbed creation tool based on GENESIS are going to be developed and provided to the tool users. The following preliminary specification for the testbed generation tool is based on parts of the GENESIS tool API. The specification forms the basis for further developments of the tool within the scope of the testbed Task 1.5. As such, this section serves as a guide to the design and development of the testbed infrastructure tool, and describes the implementation work conducted for the first extension provided with this deliverable.

3.2.1 RESTful Service generation Plug-in

This extension to the GENESIS testbed framework defines a new plug-in, which extends the available functionalities of the tool by allowing users to define testbeds containing a mixture of WSDL and RESTful Web Services. The plug-in needs to create suitable web resources that react to HTTP commands as required by the description of the RESTful Service. For example, a GET on a resource should provide a (serialized) description of the resource, while POST will update the resource accordingly.

The main functionality for this plug-in should be the creation of RESTful service mocks, based on the configuration parameters for such a service, in a similar way to the creation of WSDL Services in GENESIS. In order to achieve this functionality, the configuration file for a specific testbed needs to include a new set of elements, which define the resources, methods and parameters of the RESTful service. Instead of defining a completely new schema for this, the RESTful Service plug-in will reuse the WADL specification [4], in order to define the necessary elements of a RESTful Web Service. The schema for the testbed configuration itself only needs a single new element to link a service template to a specific WADL definition, which can be external to the testbed configuration itself. Please note that while SOA4All in general does not regard WADL descriptions, due to several severe limitations of those descriptions as related to the overall objectives of the project, the creation of mocks for RESTful APIs has different requirements and can be based on the simple definition of such APIs via WADL.

After linking a RESTful service to the overall testbed configuration, the WADL style definition then defines the basic key items of the service:

- Services - corresponding to Interfaces for WSDL, in a WADL this corresponds to the root application element. A service contains an arbitrary number of hierarchically organized resources.
- Resources - define an addressable (URI) item that can be parameterized using a number of parameter mechanisms. Resources are accessed using standard HTTP methods and can be made available in any number of representations, for example XML, JSON, PDF, etc. In addition, resources can contain child resources which will inherit parameter and path information from their parent(s).
- Methods - A resource in WADL/REST is accessed through a number of methods. A method is defined by the HTTP method it uses, as well as applicable headers and parameters. The response to accessing a resource through a method is usually a representation of the invoked resource.
- Requests - The request element defines the concrete instance of a request for a method. Resources and methods define named parameters (with default values where applicable), while requests instantiate these parameters.
- Parameters - can be defined on both the resource and method level.
- Representations - are used to define the content of a request or response, i.e. by defining the concrete media type and referring to a concrete type, defined in XML, JSON, PDF etc.

As this service description is based on the current specification of the WADL language, the users of the testbed can also utilize the WADL2Java tool[3], which allows users to quickly generate client stubs for the Web Service API defined by the WADL document.

The necessary extensions to the GENESIS testbed configuration schema have been included in Annex A of this deliverable. The tool itself was also extended by providing parsing capabilities for the extended schema, and the referenced WADL documents. Finally, a mocking component for such services was developed, which behaves in a similar manner as the mocking functionalities currently available for SOA testing tools such as SoapUI[4]. In Section 3.3, several service templates are defined, which can be used by the creators of

[3] Available at https://wadl.dev.java.net/wadl2java.html

[4] See http://www.soapui.org/userguide/mock/index.html for further information

testbed configurations (e.g., testers, or Use Case developers), to create their own testbeds based on both WSDL and RESTful services.

3.3 Service Templates

The following sections describe the templates used to instantiate test services for the generation of testbeds, suitable for testing and evaluation tasks.

A complete example testbed configuration, based on these templates and the WP9 services, as described in Section 2.3, can be found in Annex B of this document.

3.3.1 WSDL Services

Listing 1 below shows an example for a Web Service template, describing the available operations, input and output parameters and orchestration behaviour of a WSDL based Web Service.

```
<servicetemplates>
  <service name="getAndCheckServiceTemplate" type="WSDL">
    <deploy>
      <behavior>
        <!-- empty -->
      </behavior>
    </deploy>
    <undeploy>
      <behavior>
        <!-- empty -->
      </behavior>
    </undeploy>
    <operation name="getAndCheck" >
      <!–– over ride default parameters ––>
      <parameters qos_processingtime="1000"/>
      <input>
        <name type="string"/>
      </input>
      <output type="somestructure"/>
      <behavior>
        (
          InvocationPlugin."return=dbService.getData(arg.name)"
        ->
          InvocationPlugin."checkService.checkData(return)"
        )
      </behavior>
    </operation>
  </service>
</servicetemplates>
```

Listing 1: Service template for a WSDL based WS

In Annex B of this deliverable, a comprehensive service configuration for a testbed for WP9 is shown. Further testbed configurations will become available as the Use Cases continue to provide service examples.

3.3.2 REST Services

The following listing presents a fragment from a testbed configuration file. The concrete service template shows a link to a RESTful service description, given in the WADL specification shown below. For the RESTful Service generation Plug-in, this is all the information that is needed to generate a mock REST service.

```
<servicetemplates>
    <service name="newsSearchServiceTemplate" type="REST">
      <definition href="NewsSearchService.wadl">
    </service>
</servicetemplates>
```

Listing 2: Service template for a RESTful WS

As currently the Use Cases are based on mostly WSDL based services, a concrete example for a RESTful service template has been added, based on the planned usage of the eBay REST API in WP9. Listing 3 below shows the actual definition of the resources for a specific fragment of the eBay API.

```
<resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
  <resource path="newsSearch">
    <method name="GET" id="search">
    <request>
      <param name="appid" type="xsd:string" style="query" required="true"/>
      <param name="query" type="xsd:string" style="query" required="true"/>
      <param name="type" style="query" default="all">
        <option value="all"/>
        <option value="any"/>
        <option value="phrase"/>
      </param>
      <param name="results" style="query" type="xsd:int" default="10"/>
      <param name="start" style="query" type="xsd:int" default="1"/>
      <param name="sort" style="query" default="rank">
        <option value="rank"/>
        <option value="date"/>
      </param>
      <param name="language" style="query" type="xsd:string"/>
    </request>
    <response>
      <representation mediaType="application/xml" element="yn:ResultSet"/>
      <fault status="400" mediaType="application/xml" element="ya:Error"/>
    </response>
    </method>
  </resource>
</resources>
```

Listing 3: Resource definition for a RESTful WS

The Use Case work packages can create additional testbed configurations based on these templates and the different application scenarios planned for current and future prototypes.

4. SOA4All Build Environment

This section gives an overview about the SOA4All Build environment, which is used as a fundamental element in the SOA4All development process. The next subsection will give a short overview about the build process used in SOA4All. Afterwards, the testing facilities of the build system will be described showing how the build system is used to provide an appropriate foundation in the SOA4All test environment.

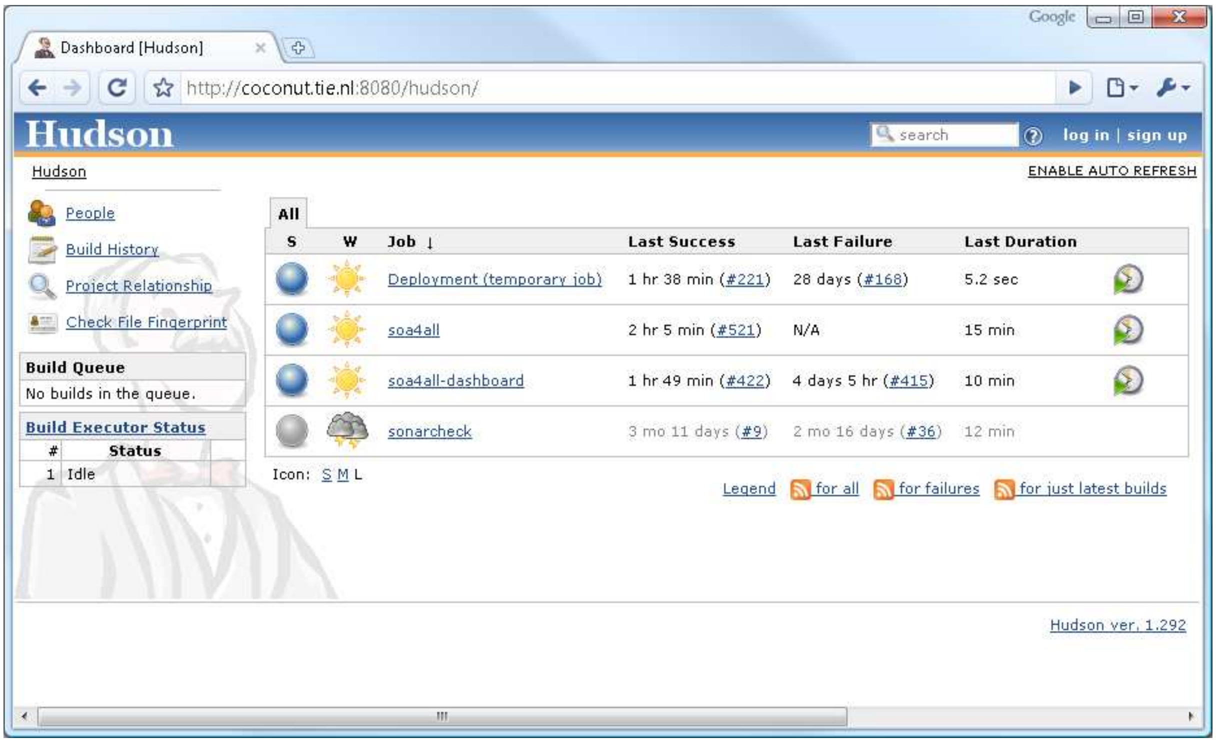
4.1 Continuous Integration Build System

The task 1.5 team has introduced the continuous build process with an automatic build tool in order to optimize the release process and to find bugs easier and faster.

Instant Notifications

In this concept, a new version of the SOA4All components is created automatically as soon as one of the developers commits new code into the code repository. This allows the SOA4All team to detect critical compilations problems immediately. Within the first months, this build process has dramatically reduced the time to find problems in the code.

The HUDSON system allows users to get informed via email whenever a build is failing. Users may also subscribe to RSS feeds allowing them to be notified as soon as problems appear. Each developer will therefore instantly see if her/his code is failing. The 1.5 team has realized this continuous build integration using the HUDSON system, which is an open source solution available at https://hudson.dev.java.net/.



Build Queue
No builds in the queue.

Build Executor Status

#	Status
1	Idle

All

S	W	Job ↓	Last Success	Last Failure	Last Duration
●	☀	Deployment (temporary job)	1 hr 38 min (#221)	28 days (#168)	5.2 sec
●	☀	soa4all	2 hr 5 min (#521)	N/A	15 min
●	☀	soa4all-dashboard	1 hr 49 min (#422)	4 days 5 hr (#415)	10 min
●	☁	sonarcheck	3 mo 11 days (#9)	2 mo 16 days (#36)	12 min

Figure 4 : The SOA4All build system

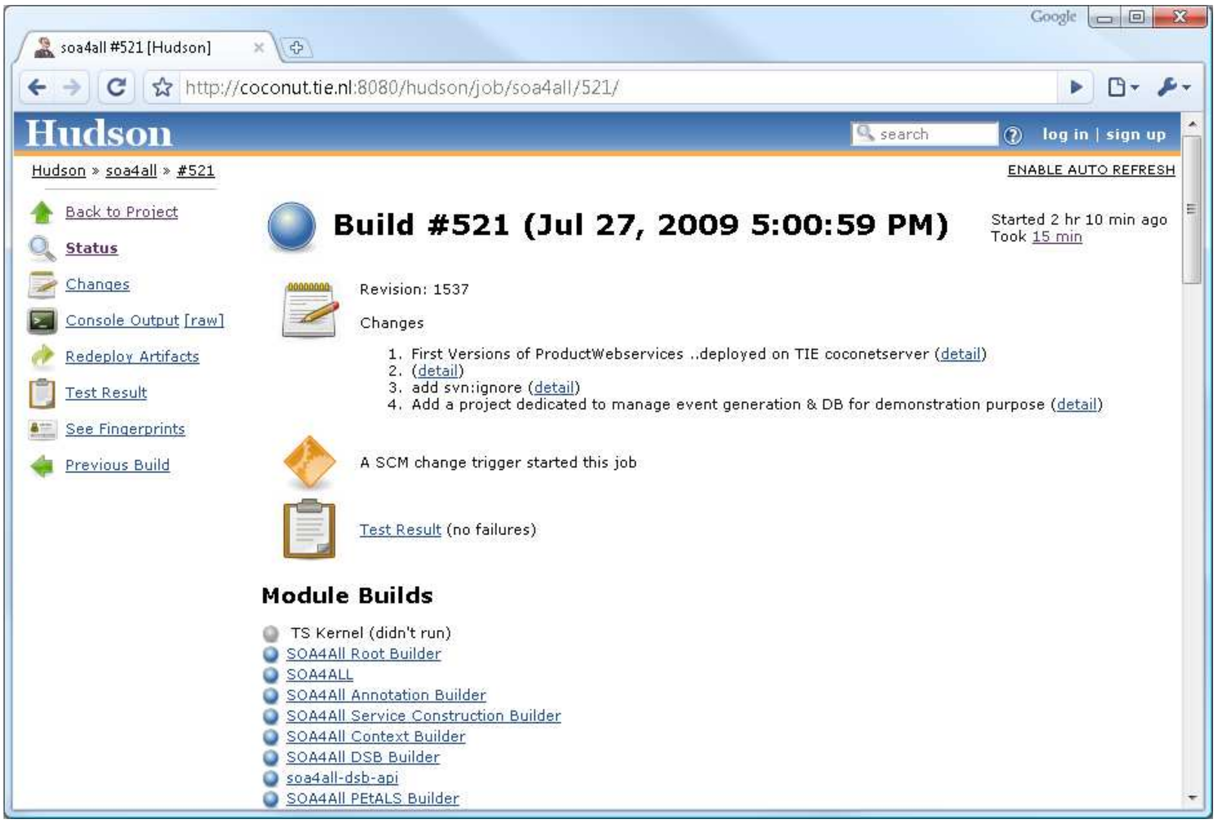
Always Up-to-Date

In addition to this, users get the capability of downloading the latest version of SOA4All whenever they want without having to wait for static release cycles. They can instantly see the changes in SOA4All and they can be sure to work with the latest achievements of the team.

Furthermore, this automated build process has allowed the team to install the latest version of SOA4All automatically on a public server (http://coconut.tie.nl:8080/soa4all). This server is updated immediately after each checkin, meaning that it always contains the latest version without any manual steps needed to update the SOA4All application.

Checkin Overview

For keeping an eye on changes, the SOA4All build system allows users to see a list of checkins and changes. This allows developers to see what has been changed and who has performed those changes. In addition to this, developers can even see different steps of the compilation by zooming into the console log of the compilations. The following screenshot shows a list of changes performed in the last checkin at the top of the image.



Build #521 (Jul 27, 2009 5:00:59 PM)

Started 2 hr 10 min ago
Took 15 min

Revision: 1537

Changes

1. First Versions of ProductWebservices ..deployed on TIE coconetserver (detail)
2. (detail)
3. add svn:ignore (detail)
4. Add a project dedicated to manage event generation & DB for demonstration purpose (detail)

A SCM change trigger started this job

Test Result (no failures)

Module Builds

- TS Kernel (didn't run)
- SOA4All Root Builder
- SOA4ALL
- SOA4All Annotation Builder
- SOA4All Service Construction Builder
- SOA4All Context Builder
- SOA4All DSB Builder
- soa4all-dsb-api
- SOA4All PEtALS Builder

Figure 5: Checkins and Modules

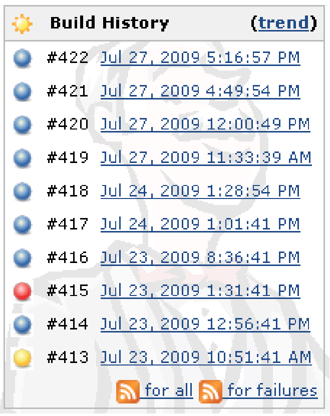
Modular Environment

The continuous build environment supports the handling of different maven targets (http://maven.apache.org) which has allowed the development team to split the SOA4All application into a consistent set of different sub parts. This modular development has allowed the SOA4All team to handle the complex project by dividing it into different concerns, which are handled by different partners and development groups. The build environment shows those different modules and their compilation and automatically combines them into different applications. For example, the main Studio module is “SOA4All Dashboard - Main”, which provides a WAR file with the SOA4All Studio, including the Dashboard and most of the

SOA4All applications used by end users.

Build History

In case that a compilation is failing or in case that a prototype does not behave as it should, the SOA4All build environment allows developers to go back in time and to download earlier compilations. In its current configuration the 1.5 team keeps the last 10 builds of SOA4All. A simple colour system shows developers if a build was successful (blue), unstable (yellow) or unsuccessful (red).



Build History (trend)

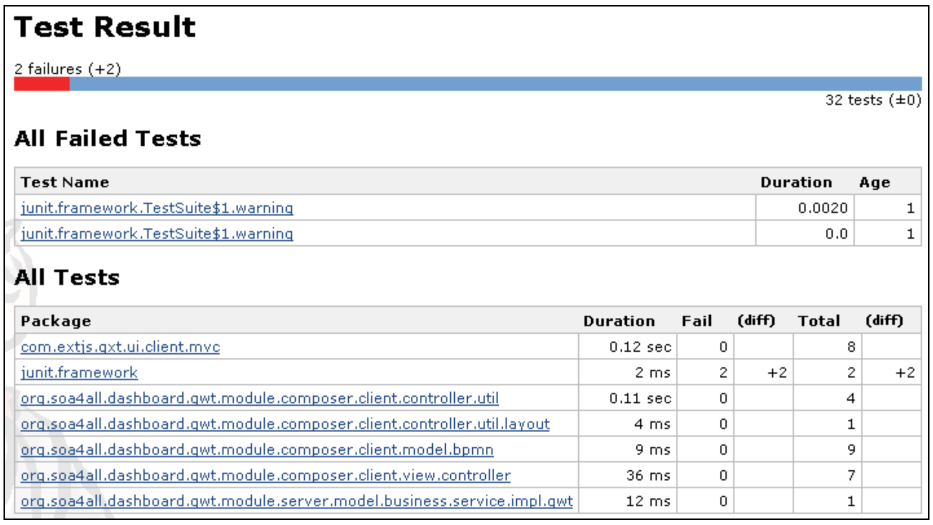
●	#422	Jul 27, 2009 5:16:57 PM
●	#421	Jul 27, 2009 4:49:54 PM
●	#420	Jul 27, 2009 12:00:49 PM
●	#419	Jul 27, 2009 11:33:39 AM
●	#418	Jul 24, 2009 1:28:54 PM
●	#417	Jul 24, 2009 1:01:41 PM
●	#416	Jul 23, 2009 8:36:41 PM
🔴	#415	Jul 23, 2009 1:31:41 PM
●	#414	Jul 23, 2009 12:56:41 PM
🟡	#413	Jul 23, 2009 10:51:41 AM

for all for failures

Figure 6: Build history

4.2 Testbed Build Targets

The SOA4All HUDSON build environment allows the SOA4All team to run tests automatically. Those tests ensure that certain functionality can be executed correctly and can therefore be seen as a first step towards ensuring the quality of the SOA4All code. The following screenshot shows a test result overview of one sub-module of SOA4All. It shows that 30 tests have been passed successfully while two tests have failed. As of July 2009, the SOA4All team has defined 184 tests in total, which are executed automatically after each build. Currently these tests are mainly comprised of unit tests (created with the jUnit testing framework). Additional tests – including component and integration tests, using the testbeds – will also be integrated with these automated build targets.



Test Result

2 failures (+2)

32 tests (±0)

All Failed Tests

Test Name	Duration	Age
junit.framework.TestSuite$1.warning	0.0020	1
junit.framework.TestSuite$1.warning	0.0	1

All Tests

Package	Duration	Fail	(diff)	Total	(diff)
com.extjs.gxt.ui.client.mvc	0.12 sec	0		8	
junit.framework	2 ms	2	+2	2	+2
org.soa4all.dashboard.gwt.module.composer.client.controller.util	0.11 sec	0		4	
org.soa4all.dashboard.gwt.module.composer.client.controller.util.layout	4 ms	0		1	
org.soa4all.dashboard.gwt.module.composer.client.model.bpmn	9 ms	0		9	
org.soa4all.dashboard.gwt.module.composer.client.view.controller	36 ms	0		7	
org.soa4all.dashboard.gwt.module.server.model.business.service.impl.gwt	12 ms	0		1	

Figure 7: Test execution and test result overview

5. SOA4All Runtime Test Environment

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 present different possibilities 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.

5.1 SOA4All Deployment Plan

In order to reach Web scale with the SOA4All Distributed Service Bus, it is necessary to go beyond the current three site deployment. This step requires two distinct tasks: i) determining the dimension of a distributed installation that allows the assumption of a Web scale deployment, and ii) plans and technicalities to scale out further upon need.

Specifying the dimensions that shall be reached in terms of deployment size is by no means an obvious task. Distribution can be shown and is necessary as soon as more than one node is considered. However, having three nodes communicating and coordinating cannot be referred to as Web scale. After all, reaching the scale of the Web with the SOA4All service infrastructure is one of the central goals of the project. For example, the FIRE experimental research facilities, presented in Section 5.2, are realized on top of several hundred nodes. Scaling up to a fraction of FIRE seems however to be adequately large for SOA4All. The objective for the end of the project is thus to install the SOA4All runtime on approximately one hundred machines; likely virtual machines. Reaching a system deployment of this size is considered to be large enough to proof scalability in the large and to ensure Web scale of the SOA4All results.

What remains is the description of how to reach this deployment size. A flexible deployment infrastructure that shall host the SOA4All runtime and platform services is depicted in Figure 8. The Distributed Service Bus is established as the connection of distributed bus nodes and semantic space nodes that are installed on top of a ProActive grid. As such, the Distributed Service Bus is given as the networked sum of all ProActive nodes, and the scale is defined by the number of such nodes.

In order to flexibly scale out, there are thus means required to install and run ProActive instances on multiple machines or virtual machines. Cloud computing infrastructures such as Amazon’s EC2 (http://aws.amazon.com/ec2/) or the newly installed Open CirrusTM Cloud Computing Research Testbed (https://opencirrus.org/) are accepted solutions in this respect. In particular EC2 established itself as the current assumed standard in most industry settings. The primary goal of the deployment plan is thus to specify the requirements and possibilities to bring the SOA4All Distributed Service Bus to the cloud, which offers the most flexibility in regards to adaptation and dimension of scale.

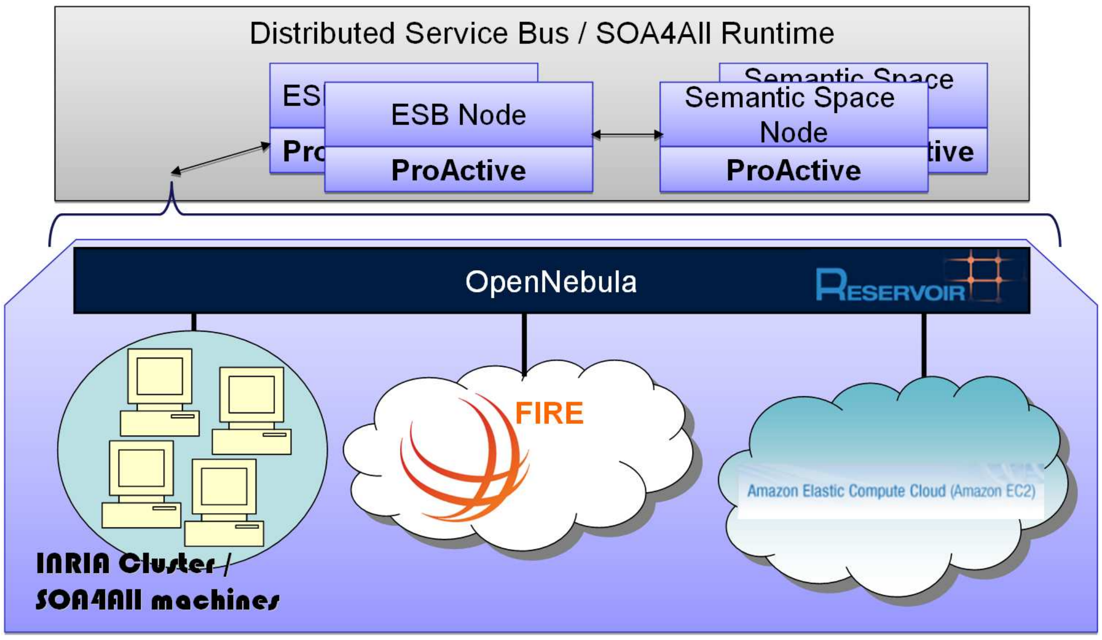

Distributed Service Bus / SOA4All Runtime
ESB Node
ProActive
Semantic Space Node
ProActive
OpenNebula
RESERVOIR
FIRE
Amazon Elastic Compute Cloud (Amazon EC2)
INRIA Cluster / SOA4All machines

Figure 8: SOA4All large-scale deployment possibilities

Less applicable in terms of scalability, however significantly more promising in regards to control, is the deployment on a closed cluster of machines. Open cloud infrastructures such as EC2 provide some resource and quality guarantees, the user has however no control or knowledge over the actual deployment and the characteristics of the virtual machines. In this respect a more limited – limited in scale – cluster provides the better means for measurements and evaluation.

The concrete approach that is suggested for SOA4All is shown in Figure 8. On the top, shaded in grey, is the Distributed Service Bus. As previously stated, this is the infrastructure established by the ProActive nodes. The deployment infrastructure presented here investigates three possibilities to scale out the number of nodes and thus to approach a Web scale installation and evaluation infrastructure:

- ProActive Cluster at INRIA Sophia Antipolis
- FIRE – Future Internet Research & Experimentation
- Amazon EC2

OpenNebula, as an open source virtual infrastructure engine that enables the dynamic deployment and replacement of virtualized services (groups of interconnected virtual machines) within and across sites, further adds to the envisaged flexible deployment infrastructure. OpenNebula extends the benefits of virtualization platforms from a single physical resource to a pool of resources, decoupling the server not only from the physical infrastructure but also from the physical location. OpenNebula is currently under investigation in the NESSI strategic project RESERVOIR (http://www.reservoir-fp7.eu/). RESERVOIR (Resources and Services Virtualization without Barriers) works towards a massive scale deployment and management of complex IT services across different administrative domains, IT platforms and geographies. In RESERVOIR breakthrough system and service technologies are developed that will serve as the infrastructure for cloud computing. It aims to achieve this goal by creative coupling of virtualization, grid computing, and business service management techniques.

The application of OpenNebula is not changing the deployment architecture of SOA4All, which is represented in Figure 8 by the grey service bus, but enables the smooth and simple delivery of elastic solutions to scaling out the infrastructure. The OpenNebula layer allows for running arbitrary further instances of the bus and semantic space software whenever needed. It offers further flexibility as OpenNebula can manage on its own different types of virtual machines. This includes for example the automatic launching of virtual machines on EC2 once no more machines are available in the controlled cluster. A further positive side-effect of this approach is the automation in deployment. Virtual instances on EC2 do not need to be started manually but this task is taken on by OpenNebula.

In summary, although we suggest three different possibilities for deploying SOA4All in the large – the proposal are described in more detail in the next section – thanks to RESERVOIR's OpenNebula virtual infrastructure engine, we are able to provide a coherent and comprehensive deployment plan across all platforms. OpenNebula allows for one implementation of the bus no matter if executed locally, or remotely or in hybrid mode, which eases the realization of Web scale.

5.2 Deployment Possibilities

In this section we discuss more deeply the three deployment approaches that are shown in Figure 8: ProActive cluster, FIRE, and Amazon EC2. As stated above, each of them can be used independently, however, in exploiting the synergies with the NESSI Strategic Project RESERVOIR and OpenNebula, it would be possible to establish an integrated deployment infrastructure.

ProActive Cluster is a 47 node cluster hosted by INRIA Sophia Antipolis for distributed deployments of ProActive based scenarios. The cluster machines are shielded from the open Internet, however, there is a dedicated front-end node installed that is accessible via SSH connections. In that way external bodies have granted access to the cluster. The machines are 2*AMD Opteron 2356 (4 cores each) with 32GB Memory and 300GB Hard Drive. They currently run Java1.6, ProActive4.1, and any additional software can be installed on demand.

As part of their involvement in SOA4All, INRIA provides access for the project to the ProActive machines. In this way, the Distributed Service Bus can be deployed on a larger number of machines for evaluation purposes. The access possibilities and available resources depend on other usage scenarios of non-SOA4All experiments and trials. The advantage however is the controlled environment, and thus, as stated previously, a more reliable and controllable testing infrastructure.

A further, more practical, advantage of applying this possibility is the target and the availability of the cluster. The cluster was established for ProActive-based distributed installations. In consequence, there are no additional needs in terms of image creation such as for example necessary for EC2, and the cluster machines already run all the necessary software basics for installing the SOA4All Distributed Service Bus. First experiments can thus be run very early in the second half of SOA4All (M13/M14).

Amazon EC2

The Amazon Elastic Compute Cloud (EC2, http:// aws.amazon.com/ec2) is a Web service which provides to the user a custom application environment on a set of distributed machines that run within Amazon's network infrastructure and data centres that guarantee a SLA commitment of 99.95% availability. The EC2 is a virtual computing environment that provides resizable (scale up or down depending on need) compute capacity. The number of machines, the application executed, and the network access (e.g., the firewall settings) can be configured dynamically through a Web service interface; the actual reservation and allocation of a physical machine is however transparent to the user.

Typically, a physical machine in the Amazon data center provides more than one instance, thus, running instances share physical resources such as network and the disk subsystem. Amazon states that “if each instance on a physical host tries to use as much of one of these shared resources as possible, each will receive an equal share of that resource. However, when a resource is under-utilized you will often be able to consume a higher share of that resource while it is available". Amazon gives minimal guarantees, but there are expected fluctuations in the availability of basic resources. Amazon EC2 is thus not the ideal infrastructure for detailed performance evaluation.

The user's application environment within the computing cloud is represented as an Amazon Machine Image (AMI). An AMI contains all the applications, libraries, data and configuration settings required for execution. In order to allow for the dynamic loading of AMIs via the Web service interface, they have to be uploaded to Amazon's Simple Storage Service (S3). Multiple AMIs with different configurations and applications can be uploaded to the S3. The user can start, monitor, and terminate as much instances of these interfaces as needed. The published files are stored in a user directory and are assigned a developer key. The bucket (storage directory) is protected with an authentication mechanism and can be made private, public or published with specific user rights.

The ProActive middleware is already deployable on EC2 and the necessary AMI is stored on S3. In order to bring the SOA4All runtime to the cloud, it will thus be necessary to install the bus software in such an existing image. Once all the necessary files and code is copied to the image, all that remains is to bundle and upload the new AMI to S3. To bundle new images there is set of EC2 script available or third party programs in order to specify the bucket to upload the image to, to determine the credentials, and to assign a full name to the image. Once the bundle is uploaded onto S3, it becomes a SOA4All personalized AMI.

The ProActive has an agreement with Amazon for several hours of free time on EC2. For initial trials, the SOA4All team will be able to profit from this special arrangement. For more detailed and larger scale realization on EC2, the consortium will buy in additional resources on a per need basis.

FIRE – Future Internet Research & Experimentation

FIRE is an initiative under the European Commission's Information and Communication Technologies research program – Challenge 1 "Pervasive and Trustworthy Network and Service Infrastructures", Objective 1.6 "Future Internet experimental facility and experimentally-driven research". It aims at a multidisciplinary research environment for investigating and experimentally validating research and developments on network and service architectures and new networking and service paradigms. FIRE seeks facilities for experimentally-driven research offering service both to academic research and industry-driven testing and experimentation. The approach chosen by FIRE is to support research at different stages of the R&D cycle, based on the design principle of "open coordinated federation of testbeds" by gradually connecting various test beds for Future Internet technologies.

In the context of the OpenNebula effort of the RESERVOIR project there are investigations ongoing to make FIRE test beds available as virtual resources under OpenNebula. This would largely ease the integration of the SOA4All environment with the FIRE testing infrastructure. Exploiting FIRE through the virtualization layer of OpenNebula would be a further added value of closer collaboration with RESERVOIR. The so-created synergies would between the two projects would ensure an integrated deployment infrastructure. Concrete details about the use of OpenNebular, in particular in regards to FIRE are ongoing work. There is certainly a need for future investigations in order to realize the envisaged plan that was presented in Section 5.1. This is work to be done in the next period of the project.

6. Conclusions

In this deliverable, we have described the different components of the testbed infrastructure environment for SOA4All. This infrastructure will be used as part of the overall efforts to evaluate SOA4All project results during the remainder of the project. The testbed infrastructure now 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.

Besides the actual configurations of the testbeds, the development of a RESTful service plug-in has been documented, which serves to extend the available functionalities for testers. Future plug-ins are going to reduce the testing effort while maintaining efficiency – one of the objectives described previously in deliverable D1.5.1. These plug-ins are going to include support for the creation of composed services and for the application of the group testing methodology to the testbeds, respectively. The development of these plug-ins is ongoing and, once new versions become available, the plug-ins will again be integrated with the overall build environment.

Finally, this deliverable described several additional efforts in the scope of Task 1.5, which were deemed necessary for a useful evaluation of project results. The overall build environment has been described and extended with testing facilities. Also the deployment of the SOA4All runtime nodes (the DSB nodes) has been planned, as a realistic environment is needed for actual results regarding scalability and performance of the developed solutions. These efforts led to the collaboration with other projects, which provide the means to test the runtime in realistic settings. Three different possibilities for deploying SOA4All in the large have been presented, but thanks to RESERVOIR’s OpenNebula virtual infrastructure engine, a coherent and comprehensive deployment plan across all the mentioned platforms will be feasible.

In the next and final deliverable of Task 1.5, we will describe the final version of the testbed environment, which will be used for the last period of the project, and will summarise the evaluation efforts and results from the experiments, which were already conducted on the testbed environment. Also, the final deployment for the SOA4All runtime nodes will be discussed, and the results of the planned scalability experiments will be reported.

7. 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. Hadley, M.J.; Web Application Description Language (WADL), Sun Microsystems Inc., Specification available at https://wadl.dev.java.net/wadl20061109.pdf
5. Lecue, F., Mehandjiev, N., Wajid, U., Namoune, A., Macaulay, L.; Deliverable D2.5.1: SOA4All Evaluation, SOA4All: Service Oriented Architectures for All - 215219.
6. Burness, A.-L., Titmuss, R., Lebre, C., Brown, K., and Brookland, A. (1999). Scalability evaluation of a distributed agent system. Distributed Systems Engineering, 6(4):129–134.
7. Jogalekar, P. and Woodside, M. (2000). Evaluating the scalability of distributed systems. IEEE Transactions on Parallel and Distributed Systems, 11(6):589–603.
8. Jiang, X., Safaei, F., and Boustead, P. (2005). Latency and Scalability: A Survey of Issues and Techniques for Supporting Networked Games. In 13th IEEE Int’l Conference on Networks, pages 150–155.
9. Shea, B. (2000). Avoiding Scalability Shock: Five Steps to Managing Performance of e-Business. Software Testing and Quality Magazine, 2(3):42–46.
10. Shalom, N. (2007). The Scalability Revolution: From Dead End to Open Road – An SBA Concept Paper. GigaSpaces Technologie.
11. 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.
12. 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.
13. 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.
14. Krummenacher, R. et al.; Towards a Scalable Triple Space; TripCom Deliverable D6.5, March 2008.

Annex A.

In order to be able to quickly generate different testbed configurations, an XSD schema for the configuration file structure was created and extended for the purposes of the RESTful plug-in. The following diagram shows the extended schema used for the GENESIS testbed configuration.

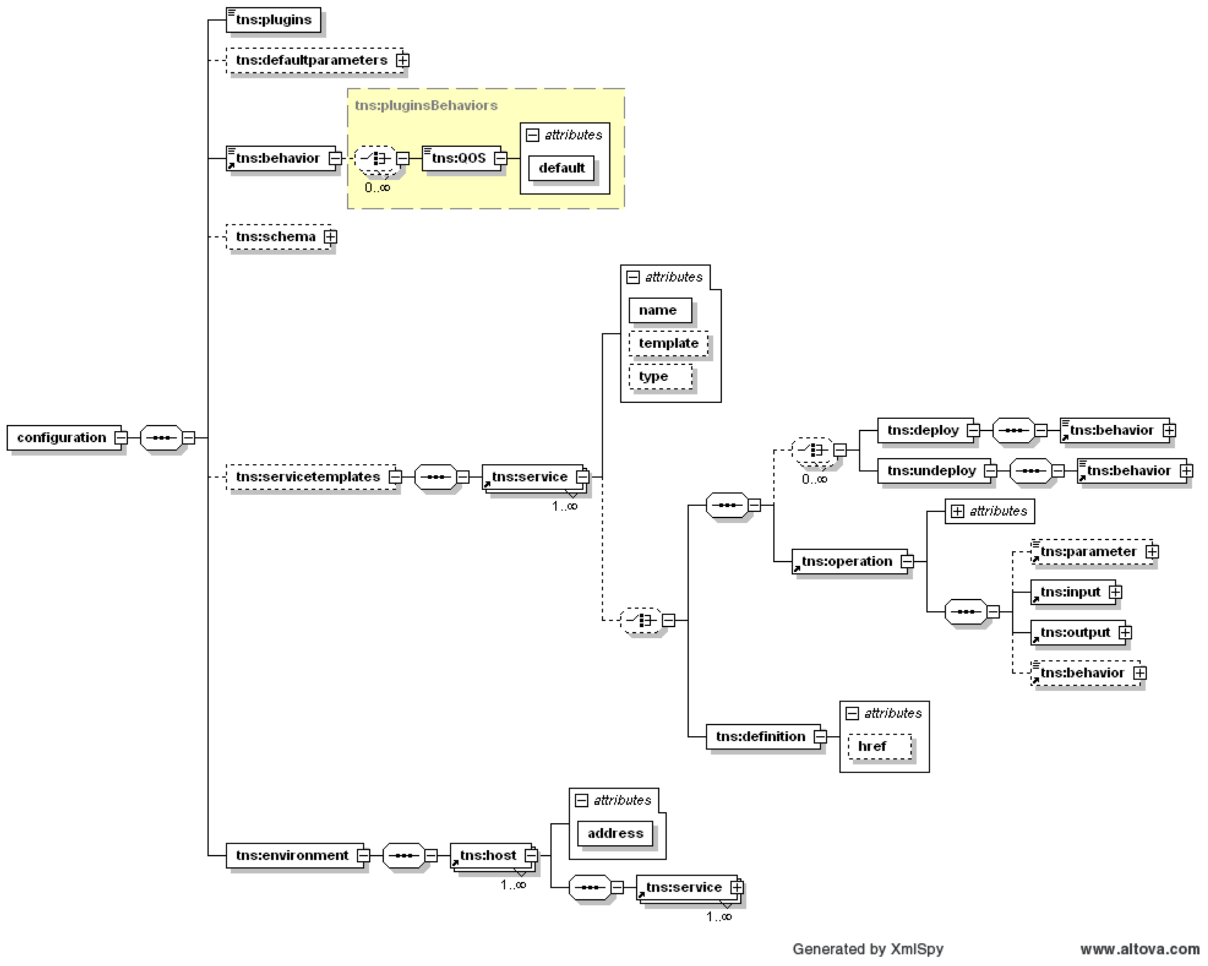


Figure 9: Schema for a GENESIS testbed configuration

The XSD version of the GENESIS testbed configuration is available in the T1.5 section of the WP1 SOA4All SVN repository.

Annex B.

The following Listing presents a concrete GENESIS testbed configuration for the WP9 scenario, as explained in Section 4 of this document. The service templates include both a WSDL based and a RESTful Web Service.

```
<configuration xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance>
  <plugins>
    at.ac.tuwien.vitalab.genesis.server.plugin.QOSPlugin
    at.ac.tuwien.vitalab.genesis.server.plugin.InvocationPlugin
  </plugins>
  <defaultparameters qos_processingtime="2000"/>
    <behavior>
      <QOS default="true">
        QOSPlugin.simulateDelay
      </QOS>
    </behavior>

  <schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:complexType name="product">
      <xs:sequence>
        <xs:element name="id" type="xs:int"/>
        <xs:element name="name" type="xs:string" minOccurs="0"/>
        <xs:element name="price" type="xs:double"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="productArray" final="#all">
      <xs:sequence>
        <xs:element    name="item"    type="tns:product"    minOccurs="0"
maxOccurs="unbounded" nillable="true"/>
      </xs:sequence>
    </xs:complexType>
  </schema>

  <servicetemplates>
    <service name="productServiceTemplate" type="WSDL">
      <deploy>
        <behavior>
            <!-- empty -->
        </behavior>
      </deploy>
      <undeploy>
        <behavior>
            <!-- empty -->
        </behavior>
      </undeploy>
      <operation name="getProductList">
        <!-- getting a product list takes 5 seconds -->
        <parameter name="qos_processingtime">5000</parameter>
        <input type="void"/>
        <output type="productArray"/>
        <behavior>
            <!-- empty -->
        </behavior>
      </operation>
      <operation name="getProduct">
```

```
        <input type="int"/>
        <output type="product"/>
        <behavior>
            <!-- empty -->
        </behavior>
      </operation>
      <operation name="getProductNameById">
        <input type="int"/>
        <output type="string"/>
        <behavior>
            <!-- empty -->
        </behavior>
      </operation>
    </service>
    <service name="eBayServiceTemplate" type="REST">
      <definition href="eBayService.wadl"/>
    </service>
  </servicetemplates>

  <environment>
    <host address="http://localhost:8070/WebServices/GeneratorService">
      <service name="productService1" template="productServiceTemplate"/>
      <service name="eBayService" template="eBayServiceTemplate"/>
    </host>
  </environment>
</configuration>
```

Listing 4: Genesis Testbed Configuration for WP9